intel®

AP-429

APPLICATION NOTE

Application Techniques for the 83C152 Global Serial Channel in CSMA/CD Mode

BOB JOHNSON
Embedded Control Applications Engineering

May 1989

Order Number: 270720-001

Information in this document is provided in connection with Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

Intel retains the right to make changes to these specifications at any time, without notice. Microcomputer Products may have minor variations to this specification known as errata.

*Other brands and names are the property of their respective owners.

†Since publication of documents referenced in this document, registration of the Pentium, OverDrive and iCOMP trademarks has been issued to Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

APPLICATION TECHNIQUES FOR THE 83C152 GLOBAL SERIAL CHANNEL IN CSMA/CD MODE

CONTENTS

INTRODUCTION

The 83C152 is an 80C51BH based microcontroller with DMA capabilities and a high speed, multi-protocol, synchronous serial communication interface called the Global Serial Channel (GSC). The GSC uses packetized data frames that consist of a beginning of frame (BOF) flag, address byte(s), data byte(s), a Cyclic Redundancy Check (CRC), and an End Of Frame (EOF) flag. An example of this type of packet is shown in Figure 1. Most 80C152 users will be familiar with UARTs, another type of serial interface. Figures 1 and 2 compare the two types of frames. The UART uses start and stop bits with a data byte between as shown in Figure 2. The 83C152 retains the standard MCS®-51 UART.

The 83C152 will be referred to as the "C152" throughout this application note to refer to the device. This application note deals with initializing and running the GSC in CSMA/CD mode only. Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is a communication protocol that allows two or more stations to share a common transmission medium by sensing when the link is idle or busy (Carrier Sense). While in the process of transmission, each station monitors its own transmission to identify if and when a collision occurs. When a collision occurs, each station involved in the transmission executes a backoff algorithm and reattempts transmission (Collision Detection). This access method allows all stations an equal chance to transmit its own packet and thus is referred to as a "peer-to-peer" type protocol (Multiple Access). Even in CSMA/CD mode, the user has several variations that can be implemented. Table 1 summarizes the various CSMA/CD options available. Most of these variations will be discussed in this application note.

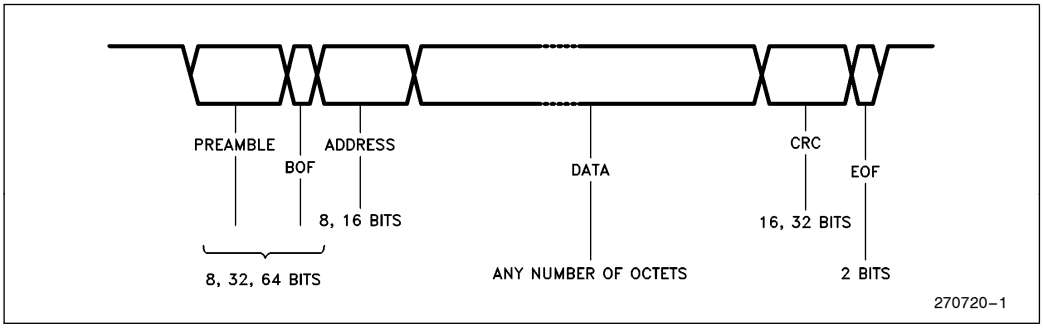

Figure 1. Packetized Frame

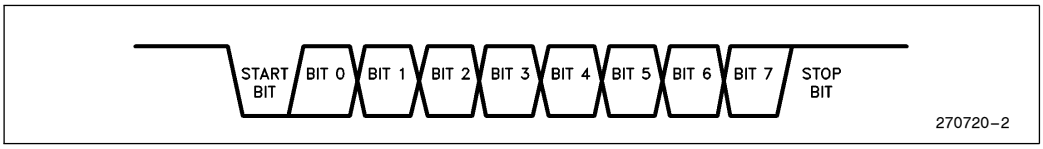

Figure 2. UART Byte

Table 1. CSMA/CD Variations Supported by C152

CSMA/CD Parameter	Options Supported by Hardware		
Preamble	8-Bits	32-Bits	64-Bits
Acknowledgement	Hardware	Software	
Backoff Algorithm	Normal	Alternate	Deterministic
CRC	16-Bit	32-Bit	
Address Recognition	8-Bit	16-Bit	S/W Extendable
Address Masking	8-Bit	16-Bit	
Jam Type	D.C.	$\overline{CRC}$	
GSC Servicing	CPU	DMA	
Data Source (Transmitter)	External RAM	Internal RAM	SFR
Data Destination (Receiver)	External RAM	Internal RAM	SFR
GSC Interface	Direct	Buffers	
Baud Rate	1.709 KPBS (minimum)	2.062 MPBS (maximum)	
# Collisions Permitted	0 to 8		
# of Slots (Deterministic Only)	1 to 63		
Time Slot	1 to 256 B/Ts		
IFS	2 to 256 B/Ts		

In this application note initializing the GSC is covered first. Starting, maintaining, and ending transmissions and receptions will then be discussed. Included in these sections will be how interrupts are generated, the software needed to respond to interrupts, and restarting the process. There are four interrupts used in conjunction with the GSC. They are: Transmit Valid, Transmit Error, Receive Valid, and Receive Error. A complete software example is shown in Appendix A. Included in the software are comments describing what and why certain sections of code are needed.

Figures 3 and 4 are flow charts that show the entire process of using the C152 GSC under CPU or DMA control. Both flow charts begin with initialization which is described in the next section. Each step in the flow charts will be described. In general, the text combines CPU and DMA control of the GSC and discusses pros and cons of each.

These flow charts were created from lab experiments performed with the C152. The purpose of the lab experiments was to implement a CSMA/CD link, over which data could be passed from one station to another. As a source for data to transmit and a method to display the data received, two terminals were used. Connecting two terminals together would not normally be encountered in an actual application. However, connecting two terminals together provided a convenient configuration on which to develop the necessary software. Connecting two terminals also created a base from which the user could implement many different designs utilizing the software provided in Appendix A.

The final experiment consisted of two parts: 1) data received by the UART to be transmitted by the GSC and 2) data received by the GSC to be transmitted by the UART. In both cases a terminal was connected to the UART on each C152 and the GSC was under DMA control. There were eight external 120 byte buffers available. Four buffers were used to store the data received by the UART and four buffers used to store the data received by the GSC.

As data is received from the UART each byte is examined, placed in an external buffer and a counter incremented. Each byte is examined to see if it equals an ASCII "carriage return" (0DH). If a match occurs, the program assumes it is the end of a line and the end of the current buffer. Once a carriage return is detected, a line feed is added and the byte count incremented. The counter is then used to load the byte count register for the appropriate DMA channel. Once a buffer is closed it's flagged as having data available for the GSC to transmit. If the next buffer was not filled with data waiting to be transmitted by the GSC, it is made available for receiving the next line. Once the GSC transmits the entire packet the buffer is flagged as empty and available for storing new data from the UART.

When a packet is received by the GSC, the data is placed in an external buffer. When the packet ends, the

number of bytes received is calculated. The current buffer is marked to indicate that the data is ready for output by the UART. The calculated byte count is used to identify how many bytes the UART should send to the terminal. When the UART sends the proper number of bytes, the buffer is made available so that the GSC may store data in it.

This has all been subjected to limited testing in the lab and verified to work with two terminals. The software has only been developed to the point that the terminals may display each other's outgoing messages and no farther. This means that some error conditions are not resolved with the current version of the software. For instance, if two terminals transmit data at approximately the same time, both messages may be displayed, even if the received data occurs within the middle of a sentence being typed. For reasons such as this, the software and hardware presented should not be used for a production product without thorough testing in the actual application.

CPU Only

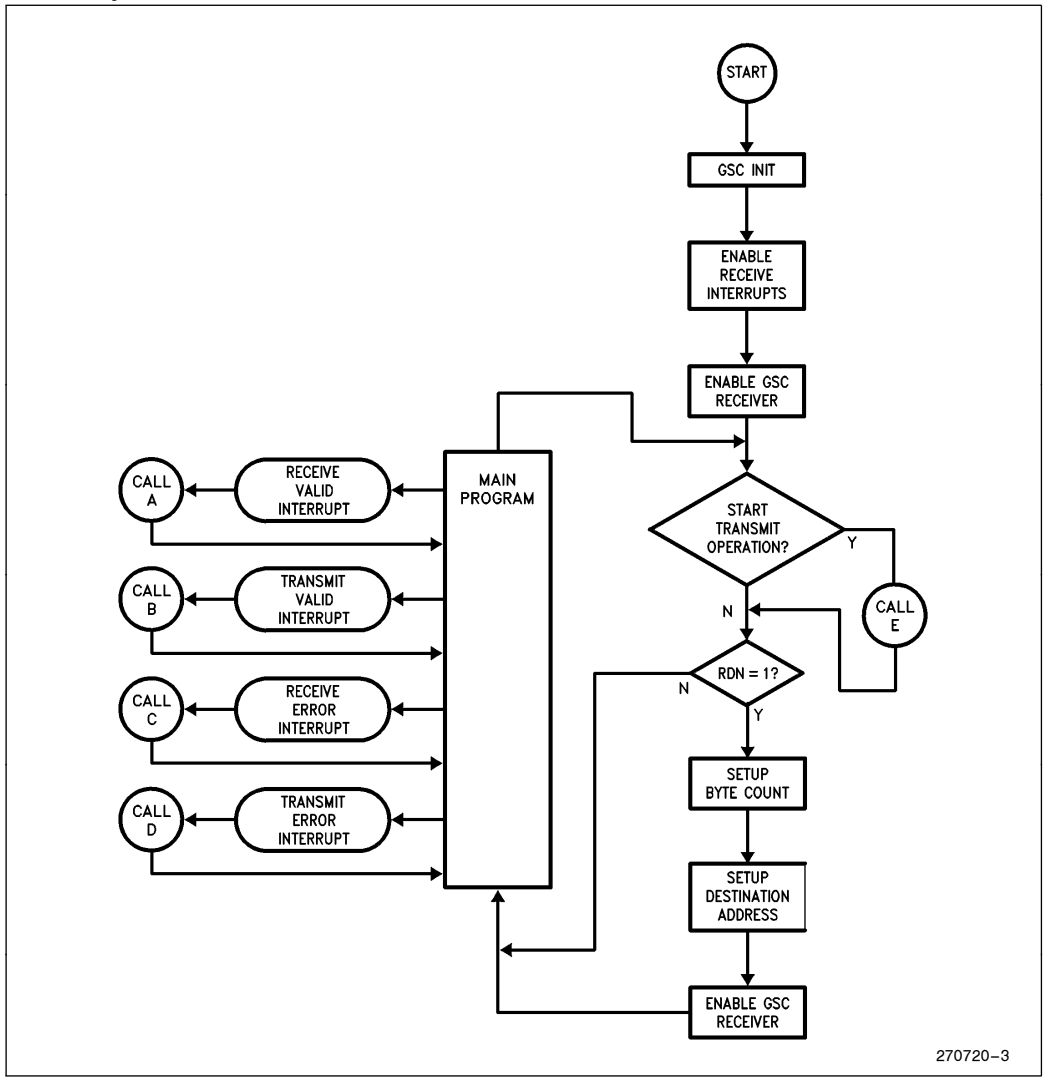

Figure 3. GSC CPU Flow Chart

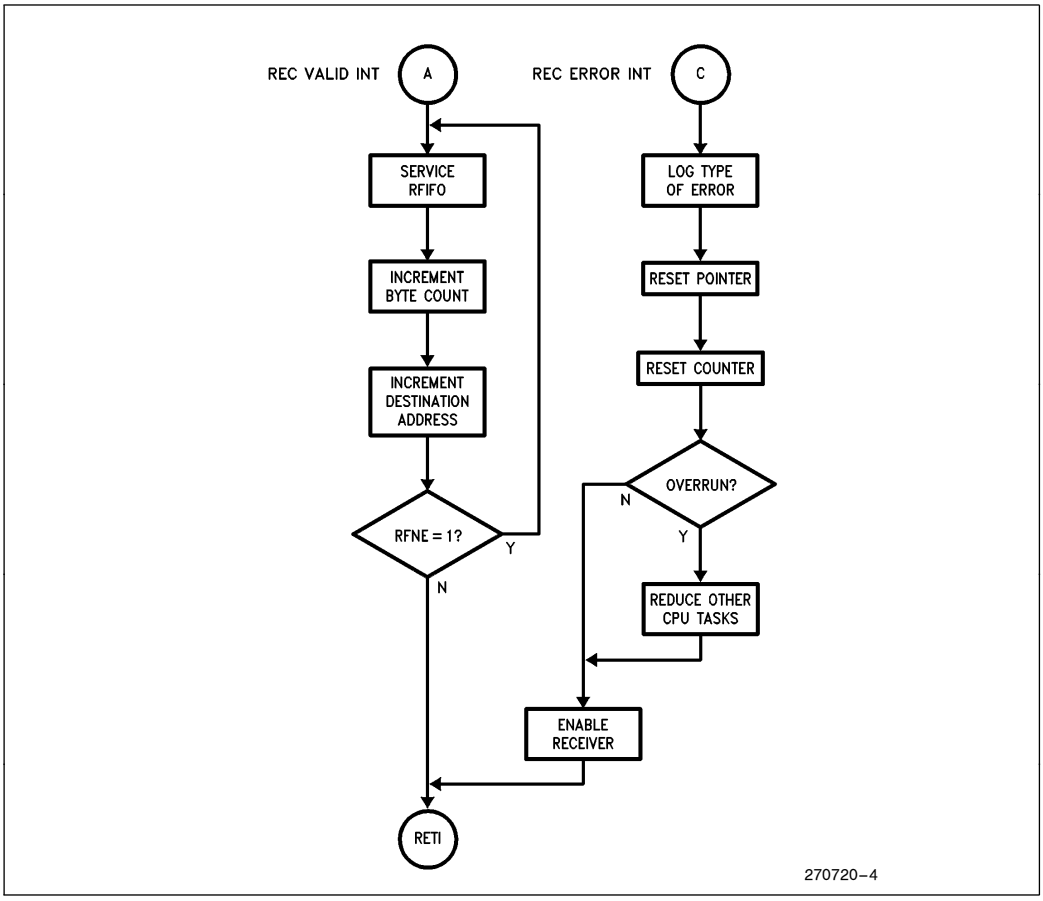

Figure 3. GSC CPU Flow Chart (Continued)

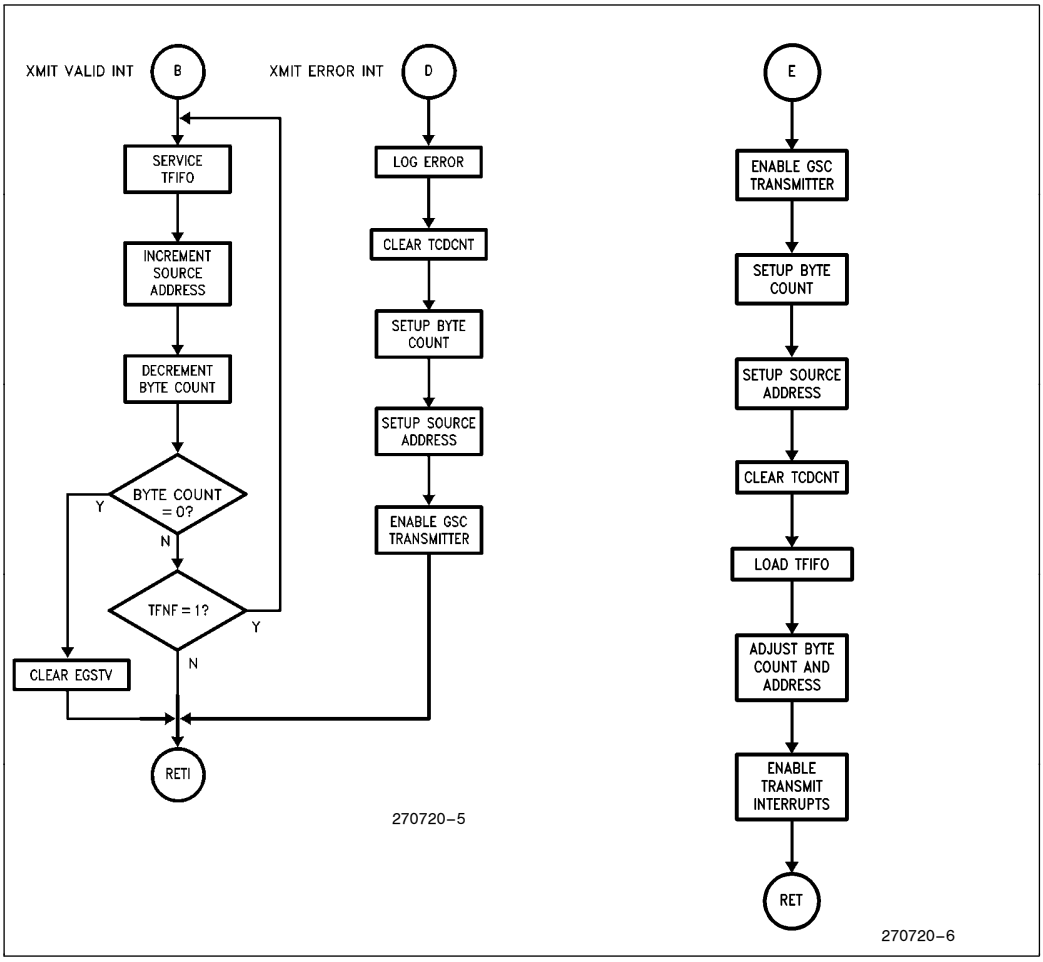

Figure 3. GSC CPU Flow Chart (Continued)

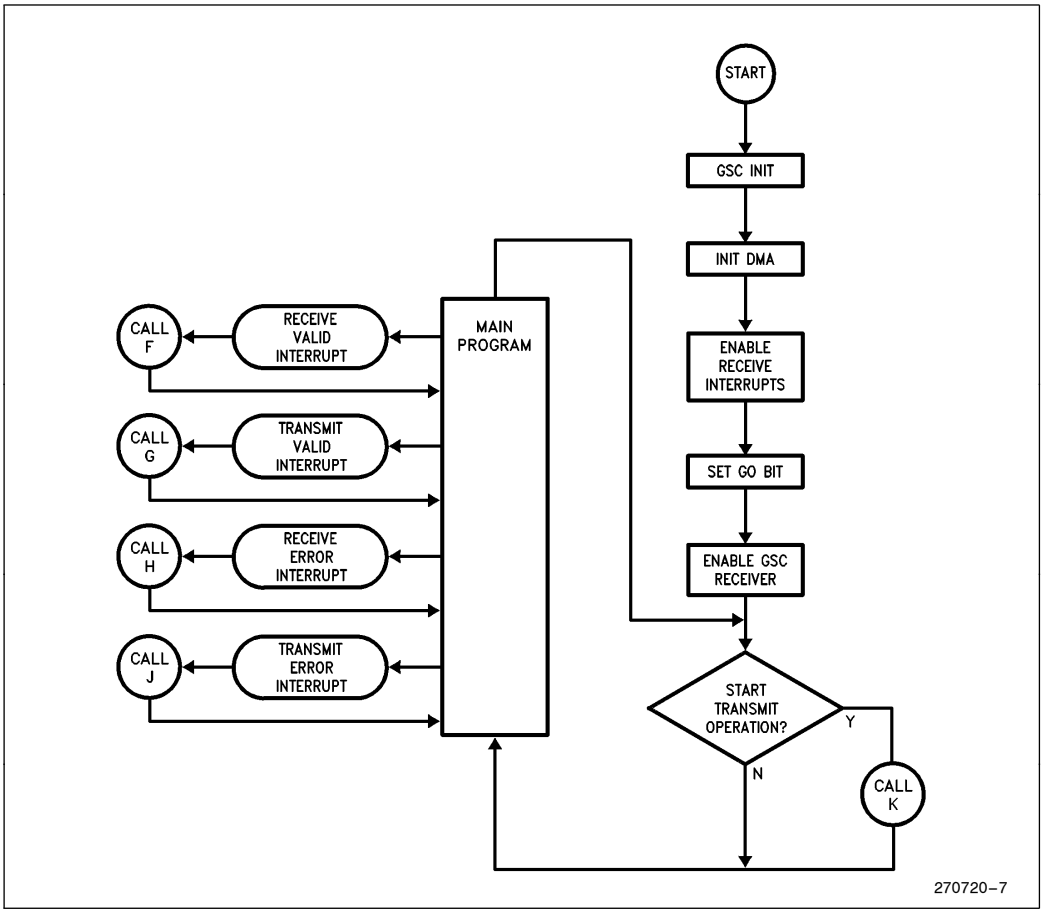

Figure 4. GSC DMA Flow Chart

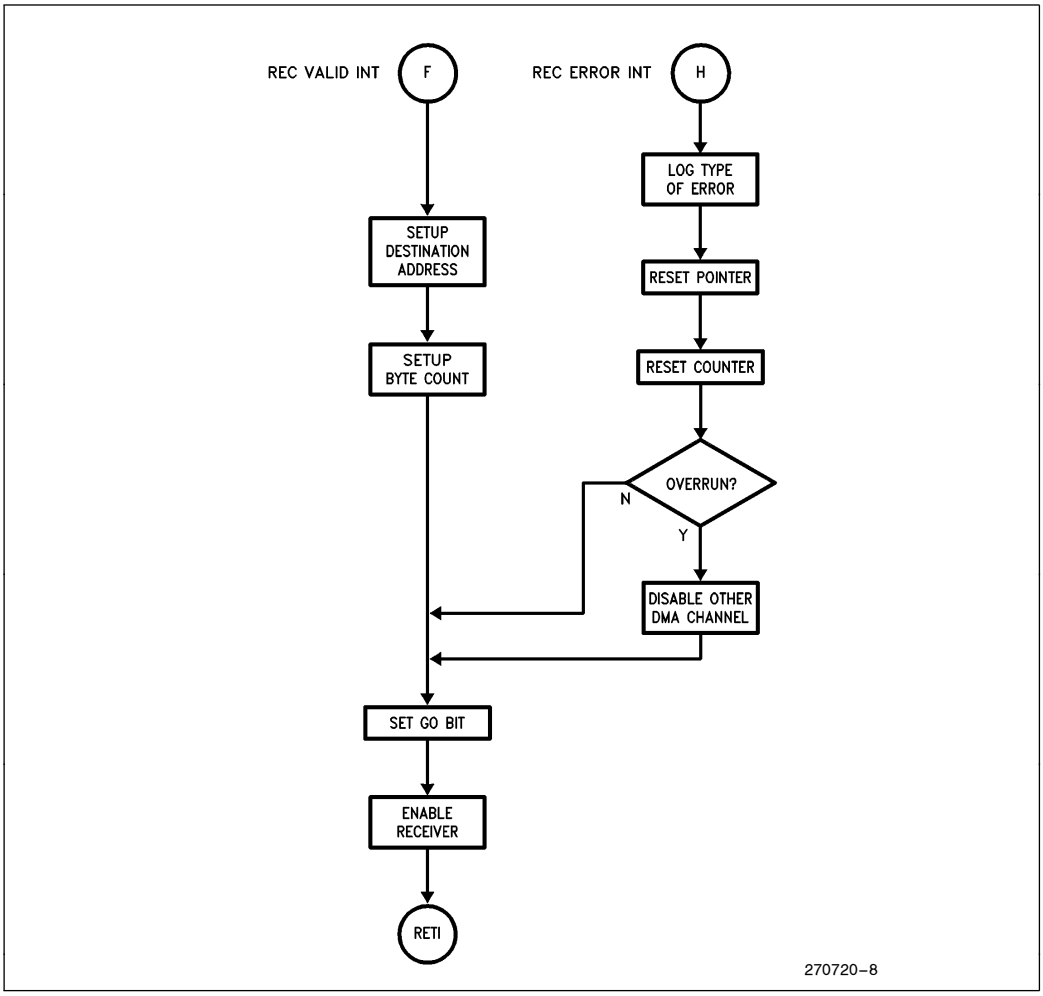

Figure 4. GSC DMA Flow Chart (Continued)

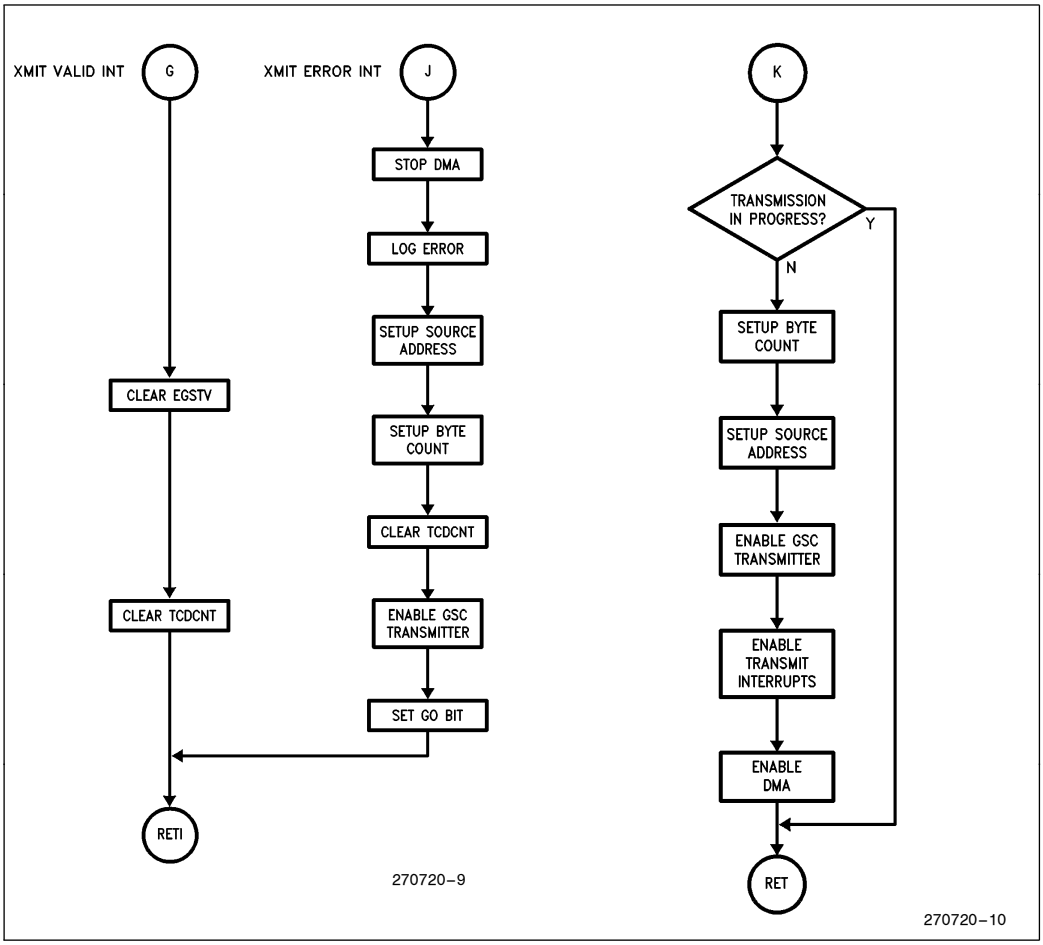

Figure 4. GSC DMA Flow Chart (Continued)

GSC INITIALIZATION

During initialization, user software sets up the hardware in the GSC so that communication may begin and institute the parameters specified by the protocol. This can further be sub-divided into two more sections. The first deals with those items which will vary according to the protocol being implemented, referred to as protocol dependent. The second section deals with those items that need to be accomplished in the same manner regardless of the protocol and are referred to as protocol independent. Table 2 shows those items of initialization which are protocol dependent. Once set up, the items in Table 2 do not have to be repeated when starting a new reception or transmission.

Table 2. Protocol Dependent Initialization

baud rate
preamble length
backoff mode (random or deterministic)
CRC
interframe space (IFS)
type of jamming signal used
slot time
addressing
enabling Hardware Based Acknowledge (HBA)

Table 2 introduces two new terms that previous CSMA/CD users may not be familiar with; Hardware Based Acknowledge (HBA) and Deterministic Collision Resolution (DCR). HBA is a method in which the GSC receiver hardware will acknowledge the reception of a valid frame and DCR is a collision resolution algorithm in which the user assigns a specific slot number to each station on the link. HBA will be covered in its own section, located later in this document. For a description on DCR or more information on HBA, please refer to the 83C152 Hardware Description in the 8-bit Embedded Controller Handbook (order # 270645).

Table 3 shows items which are protocol independent. All of the items in Table 3, except for determining how the GSC is controlled, will need to be repeated after each GSC operation, before a reception or transmission starts again.

Table 3. Protocol Independent Initialization

clearing the collision counter register
control of the GSC
initializing DMA (only if used)
initializing counters and pointers
enabling the receiver and receive interrupts
enabling the transmitter and transmit interrupts

INITIALIZATION (PROTOCOL DEPENDENT)

This section deals with those items which are part of initialization which vary according to the protocol being implemented. These parameters will typically be dictated by rules of the protocol or hardware environment. In addition, some parameters will vary according to the software implemented by the programmer. For instance, interframe space (IFS) is one of the parameters dependent on other software developed to implement a protocol with the C152.

BAUD RATE—When initializing the GSC baud rate there are two major considerations. The first is that the GSC baud rate can only be programmed in multiples of 1/8 the oscillator frequency when using the internal baud rate generator as shown in the formula given below. If a 1 MBPS rate is desired, the oscillator frequency must be 16 MHz or 8 MHz. This becomes less critical when the GSC baud rate is much lower than the desired oscillator frequency.

$$\text{GSC baud rate} = \frac{F_{osc}}{(\text{BAUD} + 1) \times 8}$$

$$\text{UART baud rate (Mode 3)} = \frac{(2^{\text{smod}})\,(F_{osc})}{(256 - \text{TH1}) \times 384}$$

The second major consideration only matters if the UART is used. In this case, when deciding on GSC baud rate and oscillator frequency the effect on the UART baud rate must be understood. As shown in the formula above, when using a timer in mode 3, baud rates generated for the UART are in multiples of 1/384 the oscillator frequency. This means that standard UART baud rates such as 9600, 2400, 1200, etc. and common GSC baud rates such as 2 MBPS, 1 MBPS, and 640 KBPS, cannot be reached with any single oscillator frequency. This can be worked around with methods such as externally clocking the timers. Externally clocking the GSC cannot be done when CSMA/CD is selected. For instance, the maximum oscillator frequency that can be used to achieve a standard UART baud rate of 9600 is 14.7456 MHz, which works out to a maximum GSC baud rate of 1.8432 MBPS which can be further divided down by multiples of 8. The program example in Appendix A uses these values.

To select a desired baud rate, the Special Function Register BAUD is loaded with an appropriate number according to the previously given formula. For instance:

```
MOV BAUD,#0     ;selects a baud rate
                ;of 1/8 the oscillator
                ;frequency
```

or:

```
MOV BAUD,#1     ;selects a baud rate
                ;of 1/16 the oscillator
                ;frequency
```

at the other extreme:

```
MOV BAUD,#0FFH  ;selects a baud rate
                ;of 1/2048 the
                ;oscillator frequency
                ;(7.2K @ 14.7456 MHz)
```

PREAMBLE LENGTH—A preamble serves four functions in CSMA/CD mode: to provide synchronization for the following frame, to contain the Beginning Of Frame flag (BOF), to let other stations on the link know that the link is being used, and to provide a window where collisions may occur and automatically reattempt transmission (backoff). Figure 5 shows what an eight-bit preamble would look like.

The C152 receiver will synchronize to the first transition and resynchronize on every following transition. For this reason a minimum preamble length can be used. On the C152 the minimum preamble length is 8-bits. However, due to network topography, other devices used, or the protocol being implemented, a larger number of transitions may be required. In these cases the C152 can be programmed for either a 32- or 64-bit preamble.

To select an 8-bit preamble:
GMOD = XXXXX01X

To select a 32-bit preamble:
GMOD = XXXXX10X

To select a 64-bit preamble:
GMOD = XXXXX11X

BACKOFF MODE—The C152 has three types of backoff modes: Normal Backoff, Alternate Backoff, and Deterministic Backoff. Normal backoff and alternate backoff are very similar and the only difference between them is when the slot timer begins counting time slots.

In normal backoff each station randomly chooses a slot based on the number of collisions that have previously occurred. After the idle (EOF) is detected, the interframe space timer and slot time timer begin at the same time. Since all devices are prevented from beginning a transmission during the interframe space, that amount of time is taken away from a device which has chosen slot 0. When a slot time is significantly larger than the interframe space, this should pose no problem as slot 0 will still provide a window for the device to begin transmission. There is a problem when the interframe space is larger than the slot time. In this case, if a device chooses slot 0, it will not be allowed to transmit because the interframe space has not yet expired. This decreases efficiency of the backoff algorithm and reduces bandwidth. Normal backoff should be used when the slot time is greater than the interframe space period.

In alternate backoff, after the idle is detected, only the interframe space timer begins. When the interframe space timer expires, the slot time timer begins. This results in extending the total amount of time spent in the backoff algorithm but preserves the entire amount of time for each slot that may be selected. Alternate backoff is recommended when the slot time is less than or equal to the interframe space period.

The deterministic backoff mode is a new resolution mode introduced by the C152. Deterministic backoff utilizes peer-to-peer communication while in normal transmission mode, and a prioritized or a deterministic algorithm while performing the resolution. Deterministic backoff operates by following standard CSMA rules when attempting to transmit a packet for the first time. However, if a collision is detected each station is

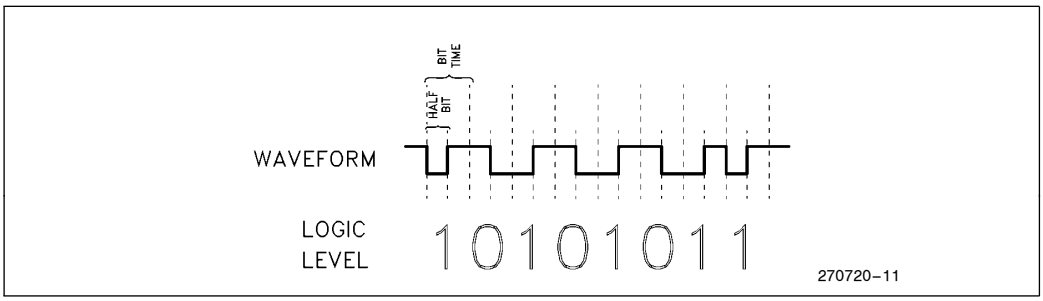

Figure 5. 8-Bit Preamble (also HBA Waveform)

restricted to only transmit during its assigned slot. The slot number is assigned by the user and up to 63 slots are available. A more detailed description on deterministic backoff is in the 80C152 Hardware Description chapter in the 8-bit Embedded Controller Handbook. Deterministic backoff is recommended if there are 64 stations or less in a network and the user wishes to remove the uncertainty that arises when using one of the other two random resolution methods already described. Another reason for using deterministic resolution is if a user wishes to assign a priority to one station's messages over that of another station's during the collision resolution period. The user should be aware that most CSMA/CD protocols that already have standards associated with them preclude the use of deterministic backoff.

To select normal backoff:
GMOD = X00XXXXX
MYSLOT = X0XXXXXX

To select alternate backoff:
GMOD = X11XXXXX
MYSLOT = X0XXXXXX

To select deterministic backoff:
GMOD = X11XXXXX
MYSLOT = X1XXXXXX

CRC—The C152 offers a choice of two types of CRC. One type of CRC is CRC-CCITT (16-bit) used in HDLC (Reference 1). The second CRC available is named AUTODIN-II (32-bit) which is used in 802.3 (Reference 2). The following formulas give the CRC generating polynomial of each.

$$\text{CRC-CCITT} = X^{16} + X^{12} + X^5 + 1$$

$$\text{AUTODIN-II} = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

The selection of which CRC to use is normally dictated by the protocol being implemented. When selecting a CRC, the user should remember that the CRC length also determines the jam time, which in turn will affect the slot time.

To select the 16-bit CRC:
GMOD = XXXX0XXX

To select the 32-bit CRC:
GMOD = XXXX1XXX

INTERFRAME SPACE—The interframe space provides a period of time for the receiver and physical medium to fully recover from a previous reception and be prepared to accept a new message. To fulfill these requirements the value programmed into IFS should be greater than or equal to the "turn around" time plus round trip propagation time. "Turn around" time is the amount of time it takes for a receiver to be re-enabled after having just received a previous packet. Calculating worst case turn around time is very complicated when the GSC is under CPU control. This is because the Receive Done bit (RDN), which signifies the end of a received packet, does not generate an interrupt. The user is required to periodically poll Receive Done to ascertain when incoming packets are complete. Since the polling sequence is sometimes altered by interrupts, these delays must also be taken into account when deciding what interframe space will be used. As an alternative, the user could choose to set-up a timer that will periodically poll the receive done bit and give a more reliable idea of what the turn around time will be. This will require that the timer interrupt be assigned a higher priority than any of the other interrupts. Since the RDN bit will be set approximately two bit times after the last CRC bit is received, in some situations it is possible to add a delay to a receive valid interrupt and check Receive Done just prior to leaving the routine. As a last resort a user could ignore the maximum response time and instead pick a number that works most of the time. The only negative result of doing this is that some frames may be missed. If acknowledgements are used, that frame would be retransmitted. However, if acknowledgements are not used, the data would be lost forever.

The programming quantum for interframe space is in bit times where a bit time is equal to 1/baud rate. The only hardware restrictions the C152 places on interframe space is that the number programmed must be even and the maximum value is 256 bit times. Other than that, the user can decide what interframe space value will be used. The interframe space should be the same for all stations on any given network.

To program the interframe space:
IFS = nnnnnnn0

where nnnnnnn0 = number of bit times programmed by the user.

The following two examples show the actual code the C152 will execute in response to a receive interrupt. Only those portions of the code associated with servicing the interrupt are shown. Added to this software, on the left edge, is the number of machine cycles it takes to execute each instruction. With this extra information the required interframe space can be calculated by totaling the number of machine cycles.

The first example gives the flow used for a valid GSC reception and the other example shows the steps taken to service an invalid reception. These examples were created by first implementing a working prototype. Once completed, the software used to service the appropriate interrupt was pulled out, selecting the worst case (longest) flow. Finally, each step was sequentially pieced together to demonstrate how the application services an interrupt. These software fragments are taken from the program in Appendix A.

The total number of machine cycles it takes to service a valid reception (59 cycles) or an invalid reception (115 cycles) is also given. As shown, an invalid reception takes the longest amount of time to service. To 115 cycles we add maximum interrupt latency, which is 9 machine cycles. The total comes out to be 124 machine cycles. It should be mentioned that the typical interrupt latency in the C152 would be about 5 machine cycles. A 9 machine cycle latency can only occur if the interrupt happens during an access to an interrupt register followed by a multiply or divide instruction and assumes that the receive error interrupt is the only high priority interrupt.

A bit time works out to be 8 oscillator periods (BAUD = 0) in this example. To calculate the number to load into IFS the following formula is used. "12" comes about from the 12 oscillator periods that make up a machine cycle.

$$\text{IFS} = \frac{12 \times (\#\text{ of machine cycles to service the interrupt})}{(\#\text{ of oscillator periods per bit time})}$$

This works out to be:

$$(12 \times 124)/8 = 186$$

This number should have a guardband added in case minor changes must be made in the routines. Since the only other enabled interrupt is the UART, a small guardband of 10 was used. The interframe space chosen is 196.

```
(# of
machine   LOC   OBJ      LINE  SOURCE
cycles    002B            358  ORG 2BH
                          359  GSC_REC_VALID:
(2)       002B  020568    360      JMP GSC_VALID_REC
                          361
                         1680  GSC_VALID_REC:
(2)       0568  C082     1682      PUSH DPL
(2)       056A  C083     1683      PUSH DPH
(2)       056C  C0E0     1684      PUSH ACC
(2)       056E  C0D0     1685      PUSH PSW
(2)       0570  71B0     1688      CALL NEW_BUFFER2_IN
                         1689
                         1031  NEW_BUFFER2_IN:
(2)       03B0  207343   1064      JB GSC_IN_MSB,GSC_IN_2
                         1067
                         1168  GSC_IN_2D_2A:
(2)       03F6  20721E   1170      JB GSC_IN_LSB,GSC_IN_2
                         1172
                         1173  GSC_IN_2D:
(2)       03F9  2074F6   1175      JB BUF2D_ACTIVE,BUFFER
(2)       03FC  758200   1179      MOV DPL,#LOW (BUF2C_ST
(2)       03FF  758303   1180      MOV DPH,#HIGH (BUF2C_S
(1)       0402  C3       1184      CLR C
(1)       0403  7476     1186      MOV A,#(MAX_LENGTH) -
(1)       0405  95F2     1192      SUBB A,BCRL1
(2)       0407  F0       1194      MOVX @DPTR,A
(1)       0408  D275     1197      SETB BUF2C_ACTIVE
(1)       040A  D272     1202      SETB GSC_IN_LSB
(1)       040C  D273     1203      SETB GSC_IN_MSB
(2)       040E  757981   1206      MOV GSC_INPUT_LOW,#LOW
(2)       0411  757803   1207      MOV GSC_INPUT_HIGH,#HI
(2)       0414  020432   1211      JMP NEW_BUF2_IN_END
                         1212
                         1251  NEW_BUF2_IN_END:
(2)       0432  8579D2   1253      MOV DARL1,GSC_INPUT_LO
(2)       0435  8578D3   1254      MOV DARH1,GSC_INPUT_HI
(2)       0438  75F300   1258      MOV BCRH1,#0
(2)       043B  75F278   1259      MOV BCRL1,#MAX_LENGTH
(2)       043E  22       1261      RET
                         1263
(1)       0572  439301   1693      ORL DCON1,#01
(2)       0575  D2E9     1695      SETB GREN
(2)       0577  D0D0     1697      POP PSW
(2)       0579  D0E0     1698      POP ACC
(2)       057B  D083     1699      POP DPH
(2)       057D  D082     1700      POP DPL
(2)       057F  32       1702      RETI

59 TOTAL CYCLES
```

Example 1. GSC Receive Valid Service Routine

```
(# of
machine  LOC   OBJ      LINE  SOURCE
cycles   0033            362  ORG 33H
                         363  GSC_REC_ERROR:
(2)      0033  020580    364      JMP GSC_ERROR_REC
                         365
                        1703  GSC_ERROR_REC:
(2)      0580  C082     1705      PUSH DPL
(2)      0582  C083     1706      PUSH DPH
(2)      0584  C0E0     1707      PUSH ACC
(2)      0586  C0D0     1708      PUSH PSW
                        1735  RCABT_CHECK:
(2)      0588  30EE07   1736      JNB RCABT,OVR_CHECK
                        1737
                        1744  OVR_CHECK:
(2)      0592  30EF07   1745      JNB OVR,CRC_CHECK
                        1746
                        1753  CRC_CHECK:
(2)      059C  30EC07   1754      JNB CRCE,AE_CHECK
(2)      059F  78E7     1756      MOV ERROR_POINTER,#CRC
(2)      05A1  5175     1758      CALL INCREMENT_COUNTER
                        1759
                         560  INCREMENT_COUNTER:
(1)      0275  D3        562      SETB C
(1)      0276  7F06      564      MOV R7,#6
                         565
                         566  INC_COUNT_LOOP:
(1*6)    0278  E6        568      MOV A,@ERROR_POINTER
(1*6)    0279  3400      570      ADDC A,#0
(1*6)    027B  F6        572      MOV @ERROR_POINTER,A
(1*6)    027C  18        574      DEC ERROR_POINTER
(2*6)    027D  DFF9      576      DJNZ R7,INC_COUNT_LOOP
(2)      027F  4001      578      JC COUNTER_OVERFLOW
                         588
                         589  COUNTER_OVERFLOW:
(2)      0282  22        591      RET
                         592
(2)      0281  22        587      RET
                         588
(2)      05A3  0205AA    1760     JMP REC_ERROR_COUNT_END
                        1761
                        1767  REC_ERROR_COUNT_END:
(2)      05AA  71B0      1772     CALL NEW_BUFFER2_IN
                        1773
                        1031  NEW_BUFFER2_IN:
                        1063
(2)      03B0  207343    1064     JB GSC_IN_MSB,GSC_IN_2

                        1168  GSC_IN_2D_2A:
(2)      03F6  20721E    1170     JB GSC_IN_LSB,GSC_IN_2
                        1171
```

Example 2. GSC Receive Error Service Routine

```
(# of
machine  LOC   OBJ     LINE  SOURCE
cycles                 1172  ORG 33H
                       1173  GSC_IN_2D:
(2)      03F9  2074F6  1175      JB BUF2D_ACTIVE,BUFFER
(2)      03FC  758200  1179      MOV DPL,#LOW (BUF2C_ST
(2)      03FF  758303  1180      MOV DPH,#HIGH (BUF2C_S
(1)      0402  C3      1184      CLR C
(1)      0403  7476    1186      MOV A,#(MAX_LENGTH) -
(1)      0405  95F2    1192      SUBB A,BCRL1
(2)      0407  F0      1194      MOVX @DPTR,A
(1)      0408  D275    1197      SETB BUF2C_ACTIVE
(1)      040A  D272    1202      SETB GSC_IN_LSB
(1)      040C  D273    1203      SETB GSC_IN_MSB
(2)      040E  757981  1206      MOV GSC_INPUT_LOW,#LOW
(2)      0411  757803  1207      MOV GSC_INPUT_HIGH,#HI
(2)      0414  020432  1211      JMP NEW_BUF2_IN_END
                       1212
                       1251  NEW_BUF2_IN_END:
(2)      0432  8579D2  1253      MOV DARL1,GSC_INPUT_LO
(2)      0435  8578D3  1254      MOV DARH1,GSC_INPUT_HI
(2)      0438  75F300  1258      MOV BCRH1,#0
(2)      043B  75F278  1259      MOV BCRL1,#MAX_LENGTH
(2)      043E  22      1261      RET
                       1262
(2)      05AC  439301  1774      ORL DCON1,#01
(1)      05AF  D2E9    1776      SETB GREN
(2)      05B1  D0D0    1778      POP PSW
(2)      05B3  D0E0    1779      POP ACC
(2)      05B5  D083    1780      POP DPH
(2)      05B7  D082    1781      POP DPL
(2)      05B9  32      1783      RETI

115 TOTAL Cycles
```

Example 2. GSC Receive Error Service Routine (Continued)

JAMMING SIGNAL—The purpose of a jam is to insure all stations on a link detect that a collision has occurred and reject that frame. To meet this need, the C152 offers two types of jamming signals. One type of jam is the D.C. jam (Figure 6) and another type is called the $\overline{\text{CRC}}$ (Figure 7) jam. A jam is forced by the TxD pin after a collision is detected but after the preamble ends if the preamble is not yet complete. The D.C. jam forces a constant logic "0" for a period of time equal to the CRC length. The $\overline{\text{CRC}}$ jam takes the CRC calculated up to the point when a collision occurs, complements the CRC, and transmits that pattern. The $\overline{\text{CRC}}$ jam should be used when A.C. coupling is used in a network. A.C. coupling normally implies that pulse transformers or capacitors are used to connect to the serial link. In these types of circuit interfaces, the D.C. jam may not be passed through reliably. One drawback of the $\overline{\text{CRC}}$ jam is that it does not always guarantee that all stations on a link will detect the jamming signal as there are no Manchester code violations inherent in the waveform. The D.C. jam is recommended whenever it can be used since this type of jam will always be detected by forcing Manchester code violations. Some protocols specify a specific type of jam signal that should be used and the user will have to decide if the C152 can fulfill those requirements.

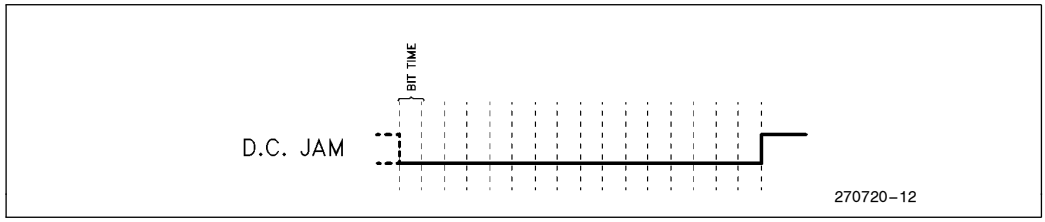

Figure 6. D.C. Jam

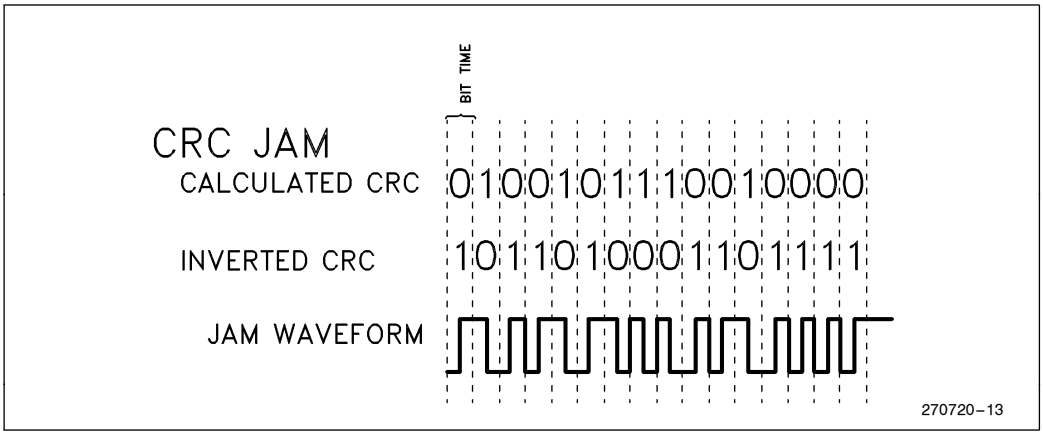

Figure 7. $\overline{\text{CRC}}$ Jam

To select D.C. jam:
MYSLOT = 1XXXXXXX

To select $\overline{\text{CRC}}$ jam:
MYSLOT = 0XXXXXXX

SLOT TIME—In CSMA/CD networks a slot time should be equal to or larger than the sum of round trip propagation time plus maximum jam time. The slot time is used in the backoff algorithm as a rescheduling quantum. The slot time is programmed in bit times and in the C152 can vary from 1 to 256.

To program the slot time:
SLOTTM = nnnnnnnn

ADDRESSING—When discussing the subject of addressing with respect to the C152, the subject should be broken down into three major topics. These topics are: address length, assignment of addresses, and address masking.

Address Length—The C152 gives a user a choice of either 8 or 16 bits of address recognition. To select 8-bit addressing the user must set the AL bit in GMOD to 0. Setting AL to 1 selects 16-bit addressing. Address recognition can be extended with software by examining subsequent bytes for a match. The only part of the GSC hardware that utilizes address length is the receiver. The receiver uses address length to determine when an incoming packet matches a user assigned address. Since transmission of addresses is done under software control, the transmitter does not use the address length bit. All bits following BOF are loaded into RFIFO, including address. The transmit circuitry is involved with addressing only if HBA is used. In this case, when HBA is selected, the transmitter must know whether or not the sending address was even or odd. Even addresses require an acknowledgement back and odd addresses do not.

When transmitting, the user must insert a destination address in the frame to be transmitted. This is done by loading the appropriate address as the first byte or two bytes of data. If a source (sending) address is also to be sent, the user must place that address into the proper position within a packet according to the protocol being implemented.

To select 8-bit addresses:
GMOD = XXX0XXXX

To select 16-bit addresses:
GMOD = XXX1XXXX

Address Assignment—When assigning an address to a station, there are several factors to consider. To begin with, there are four 8-bit address registers in the C152: ADR0, ADR1, ADR2, and ADR3. These registers are initialized to 00 after a valid reset. For this reason it is recommended that no assigned addresses should equal 0. Also, since there are four address registers, a user has a minimum of two addresses which can be assigned to each station when using 16-bit addressing or four addresses when using 8-bit addressing. Those registers not used do not need to be initialized. When using 16-bit addresses ADR1:ADR0 form one 16-bit address and ADR3:ADR2 form a second address. The C152 will always recognize an address consisting of all 1s, which is considered a "broadcast" address. An address consisting of all 1s should not be assigned to any individual station.

There are many methods used to assign addresses. Some suggestions are: reading of a switch, addresses contained in actual program code, assignment by another node, or negotiated with the system. As mentioned earlier, if HBA is being used then the LSB of the address must be 0 when acknowledgements are expect-

ed. Since more than one address can be assigned per station it is possible to use or not use HBA within the same station. This would work by assigning one address that would be even for when acknowledgements are required and another assigned address would be odd for those occasions when acknowledgements are not needed.

To assign an 8-bit address:

```
ADR0 = nnnnnnnn
```

and optionally:

```
ADR1 = xxxxxxxx
ADR2 = yyyyyyyy
ADR3 = zzzzzzzz
```

To assign a 16-bit address:

```
ADR0 = nnnnnnnn (lower byte)
ADR1 = xxxxxxxx (upper byte)
```

and optionally:

```
ADR2 = yyyyyyyy (lower byte)
ADR3 = zzzzzzzz (upper byte)
```

where xxxxxxxx, yyyyyyyy, zzzzzzzz are addresses to be assigned.

In this example there are 5 nodes (A, B, C, D, and E) with up to 4 common peripherals. The peripherals are: terminals, keyboards, printers, and modems. Assuming 8-bit addressing, a specific address bit is assigned to each peripheral: bit 1 to terminals, bit 2 to keyboards, bit 3 to printers, and bit 4 to modems. Figure 8 shows how this addressing is mapped.

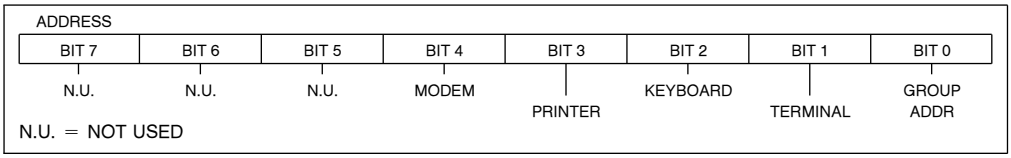

Figure 8. Group Addressing Map

Bit 0 is used to differentiate between group addresses and individual addresses. If bit 0 = 1, then the address is a group address, if bit 0 = 0, then the address is an individual address. This also complies with the HBA requirements if HBA is enabled. Table 4 defines which stations have which peripherals.

Table 4. Peripheral Assignment for Example 3

Station A:	Terminal, Keyboard
Station B:	Printer, Modem
Station C:	Terminal
Station D:	Printer
Station E:	Terminal, Keyboard, Printer, Modem

The next step is to assign each station's address and address mask. These are determined by the attached peripherals. A 1 is placed in the address register bit and address mask register bit if that station has an appropriate device. A 1 in the address register is not used since it is masked out, but will make it easier for a person not familiar with this specific software to follow the program.

	Address								Address Mask							
BIT	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
A:	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0
B:	0	0	0	1	1	0	0	1	0	0	0	1	1	0	0	0
C:	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0
D:	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0
E:	0	0	0	1	1	1	1	1	0	0	0	1	1	1	1	0

EXAMPLE 3

Address Masking—The C152 has two 8-bit address mask registers named AMSK0 and AMSK1. Bits in AMSK0 correspond to bits in ADR0 and bits in AMSK1 correspond to bits in ADR1. Placing a 1 into any bit position in AMSKn causes the corresponding bit in ADRn to be disregarded when searching for an address match.

To implement address masking:
AMSK0 = nnnnnnnn

and optionally:
AMSK1 = nnnnnnnn

where n = 1 for a "don't care address bit"
or n = 0 for a "do care address bit"

There are two main uses for the address masking capabilities of the C152. The first and simplest use is to mask off all address bits. In this mode the C152 will receive all messages. This type of reception is called "promiscuous" mode. The promiscuous mode could be used where all traffic would be monitored by a supervisory node to determine traffic patterns or to classify what information is being transferred between which nodes.

A second use of masking registers is to group various nodes together. Typically, stations are grouped together which have something in common, such as functions or location. Another term used when discussing group addresses is "multi-cast" addressing. Example #3 demonstrates how multi-cast addressing might be used.

Finally, to communicate with any station that has a printer, the address 00001001 would be sent and stations B, D, and E would receive the data. There are some limitations to using this type of scheme. Some of the more obvious are: the number of groupings is limited to the number of address bits minus 1, and it is not possible to address those stations that have a combination of attached peripherals, e.g., those stations with keyboards AND terminals. These problems can be solved using more elaborate addressing schemes.

HBA—Hardware Based Acknowledge (HBA) is a hardware implemented acknowledgment mechanism. The acknowledgement consists of a standalone preamble. An example of a preamble is shown in Figure 5. An acknowledgment will be returned by the receiver if:

- no hardware detectable errors are found in the frame
- the address is an individual address (LSB = 0)
- the transmitter is enabled (TEN = 1)
- HBA is set

An originating transmitter will expect and accept the acknowledgment if:

- HBA is set
- the receiver is enabled (GREN = 1)
- the address sent out was an individual address (LSB = 0)

If a partial or corrupted preamble is received or the preamble is not completed within the interframe space, the NOACK bit is set by the station that originally initiated transmission. HBA is a user selectable option which must be enabled after a reset.

The HBA method informs the original transmitter that a packet was received with no detected errors which saves the overhead and time that would normally be required to send a software generated acknowledgment for a valid reception. Some functions that other acknowledgment schemes implement yet are not encompassed when using HBA with a C152 is to identify packets which are out of sequence or frames which are of a wrong type.

To enable HBA:
RSTAT = XXXXXXX1

INITIALIZATION—PROTOCOL INDEPENDENT

Discussion so far has centered on those elements of initialization which will vary according to the protocol being implemented. As such, the protocol in many cases will dictate what values to use for initialization. In addition, there are some parameters set during initialization that will remain the same regardless of which protocol is being implemented. There are also some parameters which may vary for reasons other than which protocol is being used. These parameters are grouped together to form the protocol independent initialization functions. The following sections cover these elements of initialization. The discussion of initialization parameters is complete when the text covering "Starting, Maintaining, and Ending Transmissions" begins.

CLEARING COLLISION COUNTER—A transmission collision detect counter (TCDCNT) keeps track of the number of collisions that have occurred. It does this by shifting a 1 into the LSB for each collision that occurs during transmission of the preamble. When TCDCNT overflows, the C152 stops transmitting and sets TCDT. Setting TCDT signals that too many collisions have occurred and can cause an interrupt. TCDT also is set if a collision occurs after the GSC has accessed TFIFO. During normal transmission, TCDCNT can be read by user software to determine the number of collisions, if any, that have occurred. Before starting the second and subsequent transmissions, it is possible that TCDCNT already has bits shifted in from a previous transmission. This would cause TCDCNT to over-

flow prematurely. In order to preserve the full bandwidth of 8 retransmissions, TCDCNT must be cleared prior to beginning any new transmission.

To clear the collision counter:
TCDCNT = 0

CONTROL OF THE GSC—"Control of the GSC" specifies how bytes are loaded into the transmitter (TFIFO) and unloaded from the receiver (RFIFO). A user has the choice of moving data to or from the GSC under control of either user software or the DMA channels.

CPU Control—CPU control is the simplest method of servicing the GSC and allows the most control. The major drawback to CPU control is that a significant amount of time is spent moving data from the source to the destination, incrementing pointers and counters, checking flags, and determining when the end of data occurs. In addition, how the GSC interrupts function differs from when the GSC is under CPU control than when the GSC is under DMA control. Under CPU control, valid GSC interrupts occur when either RFNE (Receive Fifo Not Empty) or TFNF (Transmit Fifo Not Full) are set. The transmit error and most of the receive error interrupts still function the same regardless of which type of control is used on the GSC. The only difference in how receive error interrupts operate is that the UR (UnderRun) bit for the receiver is operational when the GSC is under DMA control. UR is disabled when under CPU control.

DMA Control—DMA control relieves the CPU of much of the overhead associated with serving the GSC and allows faster baud rates. However, the reader must realize that more details about a "yet to be transmitted packet" must be known to properly initialize the DMA channels prior to starting a transmission. In some situations, especially at high baud rates, the user must take into account DMA cycles that occur asynchronously and without any user control or knowledge. This could possibly disrupt other time critical tasks the C152 is performing. There may be no indication to a user that other ongoing tasks are being interrupted by DMA cycles taking over the bus and momentarily stopping CPU action.

When the DMA is used to service the GSC, the DMA channels will also need to be initialized and the GSC interrupts configured to operate in DMA mode. The main advantages of using DMA control is time saved and interrupts occur only when there is an error or when the GSC operation (receive or transmit) is done. This removes the necessity of continuously polling RDN and TDN bits to determine when a GSC operation is complete.

One of the most important facts to remember when deciding how to service the GSC is that unless the GSC baud rate is relatively low compared to the CPU oscillator frequency, the only method that can keep up with the receiver or transmitter is DMA control As a rule of thumb, if a user is willing to use 100% of available bandwidth of the C152 and no other interrupts are enabled besides the GSC, the maximum baud rate works out to be approximately 4.5% of the oscillator frequency. This is based on a 9 instruction cycle interrupt latency, moving a byte of data, return from interrupt and executing one more instruction before the next GSC byte is transmitted or received. At an oscillator frequency of 16 MHz, this works out to 720K bits per second. There are many steps a user could take to increase the baud rate when the GSC is under CPU control as this scenario is only a simple situation using worst case assumptions. Taking into account the amount of time available for the CPU to service the GSC as more tasks are required by the service routines or the CPU would further lower the maximum baud rate. For instance, if a user intended that GSC support only took 10% of available CPU time, this would reduce the effective baud rate by a factor of ten, making the maximum bit rate 72K. This 10% figure is an average over the period it takes to complete a frame. Situations might arise such that spurious GSC demand cycles would require much more than 10% of available time for short intervals.

INITIALIZING DMA—Since CSMA/CD is selected, it is by definition half-duplex. In half-duplex mode, only one DMA channel is needed to service both transmitter and receiver. However, it is simpler and easier to explain if both DMA channels are used. The following text is written under an assumption that both DMA channels will be used to service the GSC. Regardless of whether the DMA channel is servicing the receiver or transmitter, the DMA DONE interrupt generally should not be enabled. Also, the DMA bit in TSTAT must always be set. The GSC valid transmit and valid receive interrupts occur when RDN or TDN is set. This also eliminates a need to poll RDN or TDN to determine when a reception or transmission has ended, as is necessary when the GSC is under CPU control.

The DMA channel servicing the transmitter must have:
Destination Address = TFIFO (085H)
Increment Destination Address (IDA) = 0
Destination Address Space (DAS) = 1
Demand Mode (DM) = 1
Transfer Mode (TM) = 0

The source of data can be SFR space, internal RAM or external RAM. The byte count must be equal to the number of bytes to be transmitted, as this determines when a packet ends. TEN should be set before the DMA GO bit. It takes one bit time after TEN is set before the transmitter is enabled. The transmit valid

interrupt should be enabled after TEN is set. Since CSMA/CD is half duplex, it doesn't matter which DMA channel services the receiver or transmitter, as only one DMA channel will be active at any time.

The DMA channel servicing the receiver must have:
Source Address = RFIFO (0F4H)
ISA = 0
SAS = 1
DM = 1
TM = 0

The destination for data can be SFR space, internal RAM or external RAM. The byte count must be equal to or greater than the number of bytes to be received. Setting the byte count to 0FFFFH (64K) is one way of covering all packet lengths. GREN should be set after the DMA GO bit. The receive valid interrupt should be enabled after GREN is set. It takes one bit time after GREN is set before the receiver is enabled and for the error bits and RDN to be cleared. Before GREN is set, the user software should ensure that the RFIFO is cleared. Setting GREN does not clear the receive FIFO as stated in the hardware description.

INITIALIZING COUNTERS AND POINTERS: Whether using DMA or CPU control, pointers will be required to load the correct bytes for the transmitter and to store received bytes in their proper location. Counters are required when the GSC is under DMA control in order to keep the DMA channel active during the reception of an entire frame and to identify when a transmitted frame is to be ended. Counters are optional if the CPU is used to service the GSC, although its usefulness might be questioned.

When the GSC is under DMA control, the data pointers used are destination address registers (DARLn and DARHn) for the DMA channel responsible for the receiver and source address registers (SARLn and SARHn) for the DMA channel servicing the transmitter. The counters used are byte count registers (BCRLn and BCRHn) for the appropriate DMA channel.

The byte count for the transmitting DMA channel must be known and loaded prior to beginning actual transmission. Transmission begins when TEN and GO are set. The reason the byte count must be known prior to transmission is that when the counter reaches 0, the DMA stops loading data into TFIFO, and once TFIFO is emptied the GSC assumes a transmitted packet is complete. For the receiver the byte count can be set to the frame length if known prior to starting reception or the byte count can be set to a maximum frame packet length that will ever be received. Another alternative is to set the byte count equal to 0FFFFH. This option may be chosen if the length of received packets are totally unknown. If 0FFFFH is used, the user must make sure that there is some method to accommodate this many bytes. If maximum buffer size is a limiting factor, then that would be used.

When the GSC is under CPU control, internal RAM is typically used for pointers and counters. These pointers and counters would be updated by software for each byte that is received or transmitted. An interrupt is generated as long as there is at least one byte in the receive FIFO. An interrupt is also generated as long as there is room for one byte in the transmit FIFO. It is in the interrupt service routine that counters and pointers are updated and data is transferred to or from the GSC FIFOs. One advantage of CPU control is that the length of received or transmitted packets need not be known prior to the start of GSC activities. When the GSC is under CPU control, user software determines when a transmission has ended. For moving targets, CPU control allows the user software to determine where to store received data at the time it is transferred to RFIFO.

So far only initialization of the GSC and DMA has been explained. In order to use the GSC, the receiver, transmitter, and associated interrupts need to be enabled. These are covered in the following section.

ENABLING RECEIVER AND RECEIVER INTERRUPTS—There are two receiver interrupt enable bits, EGSRV (Receive Valid) and EGSRE (Receive Error) and one bit to enable the receiver (GREN). The interrupts should always be enabled whenever the receiver is enabled. Once this is done, a user can wait for interrupts to occur and then service the GSC receiver. The conditions which will cause the CPU to vector to GSC receiver interrupt service routines are described in the 8-Bit Embedded Controller Handbook.

In most CSMA/CD applications, GSC receivers will be enabled all the time once the C152 has been initialized. The only time the receiver will not be enabled is when a reception is completed or a receive error occurs. When this happens, the GSC receiver hardware clears GREN, which disables the receiver. The receiver must then be re-enabled by software before it is ready to accept a new frame. One way to do this when under DMA control is to set the receiver enable bit (GREN) in the receiver interrupt service routine. Similarly, the GSC receive interrupts should always be enabled and remain so except for the period of time that it takes to service an interrupt.

Once set, the GSC receiver interrupt enable bits always remain set unless cleared by user software. About the only valid reason for clearing the receiver interrupt enable bits is so that certain sections of code will not be disrupted by GSC activities. If the interrupts are disabled while the receiver is enabled, the amount of time the interrupts are disabled should not exceed 24 bit times. If the interrupts are disabled for a longer period of time, the receive FIFO may be over written.

It is a good practice to enable the GSC receiver interrupts prior to enabling the receiver when under CPU control. Another alternative is to clear the EA bit while enabling the GSC receiver and receiver interrupts. However, this could increase interrupt latency. If something like this is not done, a higher priority interrupt may alter the program flow immediately after the receiver is enabled and prior to enabling the interrupts. This in turn could cause the receiver to overflow. When the receiver is under DMA control the situation is different. First, the interrupts cannot be enabled before the receiver because if RDN is set from a previous reception, the receive valid service routine will be invoked but no reception has yet taken place. The correct sequence when under DMA control would be to set the DMA GO bit, enable the receiver, then enable the receiver interrupts. In this case the worst that could happen is a slow response to RDN getting set. Even this can be worked around by making receive valid the only high priority interrupt.

To enable the receiver interrupt enable bits and the receiver this sequence should be followed:

```
IEN1 = XXXXXX11
RSTAT = XXXXXX1X
```

or if under DMA control:

```
DCONn = XXXXXXX1
RSTAT = XXXXXX1X
IEN1 = XXXXXX11
```

ENABLING TRANSMITTER AND TRANSMIT INTERRUPTS—There are two transmit interrupt enable bits—EGSTV (Transmit Valid) and EGSTE (Transmit Error) and one transmitter enable bit—TEN (Transmitter ENable). The interrupts should always be enabled whenever the transmitter is enabled. Once this is done, a user can wait for interrupts to occur and then service the GSC transmitter. Conditions which will cause the CPU to vector to GSC transmit interrupt service routines are described in the 8-Bit Embedded Controller Handbook.

Compared with the receiver, opposite conditions exist concerning when the transmitter is operational and the sequence of enabling transmitter versus transmit interrupts. First, the transmitter and its interrupts are disabled all of the time except on those occasions when a transmission is desired. The user's application determines when a transmission is needed. Status of the message, how full a buffer is, or how long since the last message was sent are typical criteria used to judge when a transmission will be started.

When a transmission is complete, the interrupts and the transmitter should be disabled. This is particularly true for the transmit valid interrupt as TFIFO will most likely be empty and TFNF (Transmit FIFO Not Full) will be set. TFNF = 1 is the source of transmit valid interrupts when the GSC is serviced under CPU control.

The transmitter should be enabled before enabling the transmitter interrupts. If the GSC is under CPU control and the interrupts are enabled first, TFIFO may be loaded with data in response to TFNF being set. When TEN is set, data already loaded into TFIFO would be cleared. Consequently, data meant to be transmitted would be lost. If the GSC is under DMA control, it is possible that an interrupt would be generated in response to TDN being set from the previous transmission, yet no transmission has even started since the interrupts were enabled. If using the DMA channels to service the transmitter, TEN must be set before the GO bit for the DMA channel is set. If not, the DMA channels could load TFIFO with data, and when TEN is set that data would be lost.

The correct sequence to enable the transmitter and its interrupt enable bits is:

```
SETB TEN
SETB EGSTE
SETB EGSTV
```

or if under DMA control:

```
SETB TEN
SETB EGSTE
SETB EGSTV
ORL DCONn, #01
```

Once all initialization tasks shown so far are completed, reception and transmission may commence. The process of starting, maintaining, and ending transmissions or receptions is covered next.

STARTING, MAINTAINING, AND ENDING TRANSMISSIONS

Prior to starting a transmission, the user will need to set TEN. This enables the transmitter, resets TDN, clears all transmit error bits and sets up TFIFO as if it were empty (all bytes in TFIFO are lost) after a GSC bit clock occurs. Once TEN is set, actual transmission begins when a byte is loaded into TFIFO. Figure 9 is a block diagram of the GSC transmitter and shows how it functions. Once a byte has entered TFIFO, transmission begins. The first step is for the GSC to determine if the link is idle and interframe space has expired. Actually, this occurs continuously, even when not transmitting, but transmit circuitry checks to make sure these conditions exist before transmitting. If these two conditions are not met, the C152 will wait until they are. Once interframe space has expired, $\overline{DEN}$ is forced low for one bit time prior to the GSC emitting a preamble and BOF. About the time the BOF is output, a byte from TFIFO is transferred to the shift register. As bits are shifted out this register, they pass by the CRC generator, which updates the current CRC value. Bits then enter the data encoder which forms them into Manchester coded waveforms and out TxD. If TFIFO is empty when the shift register goes to grab another byte, the GSC assumes it is the end of data. To complete a frame, bits in the CRC generator are passed through the data encoder and the EOF is appended. One part of the block diagram in Figure 9 is the transmit control sequencer. The transmit control sequencer's purpose is to determine which state the transmitter is in such as Idle, Preamble, Data, or CRC. To perform this function it has connections to all circuits in the transmitter. These connections are not shown in order to make the diagram easier to read.

If the transmitter is under CPU control the first byte is loaded with user software. TFIFO should be filled and counters and pointers updated before proceeding with any other tasks required by the CPU. There is room for up to three bytes in TFIFO. Before loading the first byte, users should examine TDN to ensure that any previous transmissions have completed. If TEN is set before the end of a transmission, that transmission is aborted without appending a CRC and EOF but the interframe space will still be enforced before starting again. A user can identify when TFIFO is full by examining TFNF (Transmit Fifo Not Full). TFNF will always remain at a logic 1 as long as there is room for at least one more byte in TFIFO. There is a one machine cycle latency from when a byte is loaded into TFIFO until TFNF is updated. Because of this latency, the status of TFNF should not be checked immediately following the instruction that loaded TFIFO but should be examined two or more instructions later. Whenever TFNF is set, an interrupt will be generated if EGSTV is set. In response to the interrupt, bytes should be loaded into TFIFO until TFNF is cleared and update any pointers or counters.

Once the user is through with transmitting bytes for the current frame, the GSC transmit valid interrupt (EGSTV) should be disabled. This is to prevent the program flow from being interrupted by unnecessary GSC demands as TFNF will remain set all the time. The GSC transmit error interrupt (EGSTE) must remain enabled as transmit errors can still occur. While under CPU control there is no interrupt associated with transmit done (TDN) so a user must periodically poll this bit to determine when actual transmission is complete. After the last byte in TFIFO is transmitted there is a delay until TDN is set. This delay will be equal to the CRC length plus approximately 1.5 bit times for the EOF. The CRC is appended after the end of data by GSC hardware.

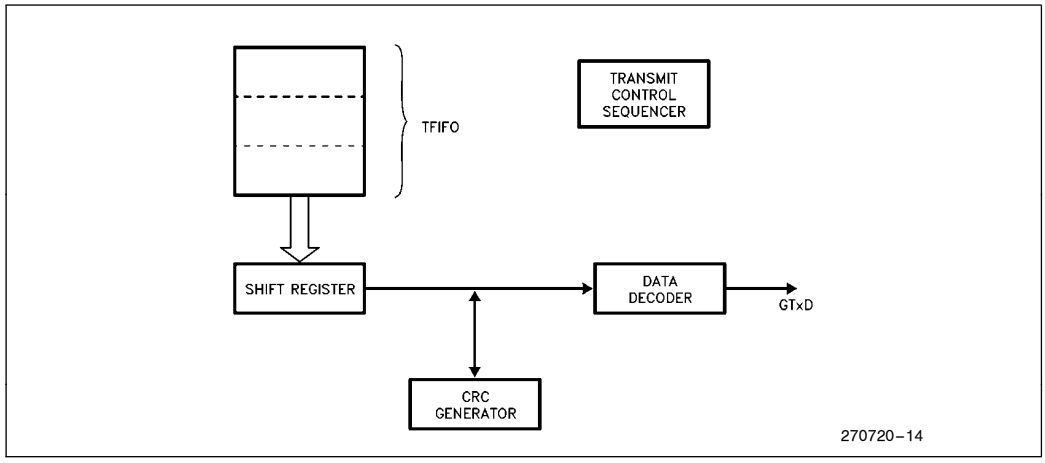

Figure 9. Transmitter Block Diagram

To start a transmission when the GSC is under DMA control, users should first enable the transmitter by setting TEN, then set the GO bit for the appropriate DMA channel. Before the GO bit is set users must initialize the GSC and DMA. Thereafter, the DMA loads the first byte that begins actual transmission and keeps the transmit FIFO full until the end of transmission. In this case, transmission ends when the byte count reaches 0, which means the length of the message to be transmitted must be known before transmission begins.

The DMA channel examines TFNF to determine when the transmitter needs servicing. When a byte is transferred into TFIFO, the DMA channel takes control of the internal bus and the CPU is held off for one machine cycle. This is the only overhead associated with the actual transmission when under DMA control. This is significantly less than the overhead associated with each byte that must be loaded by software when the GSC is under CPU control. When the DMA is servicing the transmitter, at least one machine cycle occurs between each DMA load. This prevents the DMA from hogging the internal bus when servicing the transmitter. It takes five machine cycles to load three bytes to initially fill TFIFO. When transmission ends, TDN will be set and when the GSC is under DMA control it is the setting of TDN that begins the GSC interrupt service routine.

The discussion so far assumes there are no errors during transmission of a frame. However, in CSMA/CD there is always a possibility of an error occurring and part of maintaining transmission is servicing those errors. In the C152 when an error is detected an error bit is set. At the same time the error bit is set, TEN is cleared which disables the transmitter. Types of errors that can occur are: collision detection errors (TCDT), no acknowledgement errors (NOACK) (if HBA is enabled), and underrun errors (UR) (if the DMA channels are used to service the transmitter). After setting the error bit, the C152 jumps to the transmit error vector if EGSTE (Transmit Error enable) is set. Depending on the protocol implemented, a user may wish to take some specific response to an error but in almost all cases the transmitter will be re-enabled and the same data retransmitted. This requires that counters and pointers be initialized, the transmitter enabled, and TFIFO filled. Another frequent action taken is to log the type of error for later analysis or to keep track of specific trends. Once transmission is restarted, the same flow is followed as before, as if no error occurred.

STARTING, MAINTAINING, AND ENDING RECEPTIONS

In most applications, the receiver is always enabled and reception begins when the first byte is loaded into RFIFO. Figure 10 shows a block diagram of the receiver.

As indicated in Figure 10, before the first byte is loaded into RFIFO, the address is checked for a matching address assigned by ADRn. A user can disable address recognition by writing all 1s to the address mask register(s), AMSKn. In this mode all frames with a valid BOF will be received. When the first byte is loaded into RFIFO, RFNE is set. If the address does match, there is a delay of about 24 or 40 bit times from reception of the first bit until a byte is loaded into RFIFO depending on which CRC is chosen. This is due to CRC strip circuitry and the bits required to fill up the shift register.

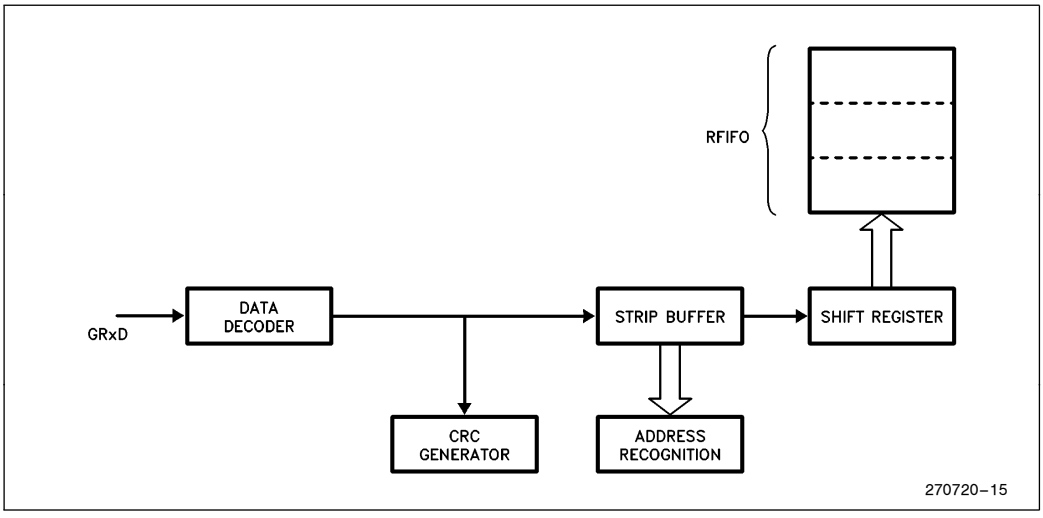

Figure 10. Receiver Block Diagram

When the GSC is being serviced by the CPU, an interrupt is generated when RFNE is set and if EGSRV is enabled. The user typically responds to an interrupt by removing one byte from RFIFO and storing it somewhere else. The user should check RFNE before leaving the interrupt service routine to see if more than one byte was loaded in to RFIFO. While under CPU control, there is no interrupt generated when reception is complete although receive done (RDN) is set. When RDN is set, the receiver is disabled and user software has to re-enable it. To determine when a frame has ended, the user must periodically poll RDN. After a frame has ended, the user will normally reinitialize pointers, reset counters, and enable the receiver. RDN will not be set when the last byte is transferred to RFIFO because the EOF will not be recognized yet. It takes approximately 1.7 bit times of link inactivity for the EOF to be recognized.

When the GSC is controlled by the DMA channels an interrupt is generated when RDN is set for a valid reception. At this point all a user needs to do is to set the source address registers, set the byte count, set the GO bit, and enable the receiver. Whenever the GSC receiver is being serviced by the DMA channels, the GO bit should be set before the receiver enable bit, GREN. This is to ensure that the DMA channel is active whenever the receiver is enabled. If the receiver is enabled before the DMA channel, it is possible that an interrupt would alter the program flow. An interrupt could delay setting the GO bit so that data is received while the DMA channel is prevented from servicing the GSC. Consequently, an overrun error occurs.

For the GSC receiver, as in the transmitter, an error is always possible. Conditions that set the error bits are the same regardless of how the receiver is being serviced. Possible errors are: receiver collision (RCABT), CRC error (CRCE), overrun (OVR), and alignment error (AE).

The only type of error that user software can take actions to prevent is an overrun error. In this case, when an overrun error occurs it is because the receiver could not be serviced fast enough. Under DMA control, the only way this could happen is if the other DMA channel prevented servicing the GSC by the DMA or the user cleared the GO bit. Solutions to these problems are to turn off the second DMA channel when receiving and not mess around with the GO bit during reception. To determine if the GSC is receiving a packet, the byte count of the appropriate DMA channel can be examined. If the GSC is under CPU control and an overrun occurs it is because there are too many other tasks the CPU is doing or the baud rate is just too high for the CPU to keep up. A solution to this problem is to either cut back on the number of tasks the CPU must perform while a packet is being received or to switch to DMA control of the GSC.

In all other cases, about all the C152 can do when a receive error occurs is to log the type of error, discard the data already received, and to re-enable the receiver for the next packet. These actions would also be taken for an overrun error.

SUMMARY

Hopefully, this application note has given the reader some insight on how to set up the GSC parameters, how to transmit or receive a packet, and how to respond to error conditions that may arise. The process of obtaining data for transmission or what to do with data received has been left open as much as possible as these vary widely from application to application. In some cases, all the data will be managed by another, more powerful processor. In this situation, the user will have to implement another interface between the main processor and the C152.

Although the whole process of using the C152 may at first, seem confusing and complicated, breaking down this process into steps may make utilizing the C152 much simpler. One suggestion of the steps to follow is:

1) INITIALIZATION
   - A) Baud rate
   - B) Preamble
   - C) Backoff
   - D) CRC
   - E) Interframe space
   - F) Jamming signal
   - G) Slot time
   - H) Addressing
   - I) Acknowledgment
   - J) Clearing the collision counter
   - K) Controlling the GSC
   - L) DMA initialization (if used)
   - M) Counter and pointer setup
   - N) Enabling the GSC
   - O) Enabling the interrupts

2) TRANSMITTING/RECEIVING PACKETS
   - A) Starting transmission/reception
   - B) Maintaining GSC operations
   - C) Ending transmission/reception
   - D) Responding to errors

These steps can be used as a checklist to ensure that the minimum set of functions have been implemented that will allow the GSC to be used in almost any application. The list also demonstrates that the bulk of the tasks the user must implement is in initializing the GSC. Once initialization is accomplished, there is comparatively little work left to implement an application.

APPENDIX A
SOFTWARE EXAMPLE

The following example demonstrates how the DMA can be used to service the GSC in a specific environment. Figure 11 shows a diagram of the hardware used. As shown, the UART is used as a source and destination for data transferred by the GSC. Also shown in Figure 11 are some DIP switches. These DIP switches determine source and destination addresses. The switches are read only once after a reset. The hardware environment is shown for informational purposes only and is not necessarily a real application that would be implemented by a user. Even so, with some minor changes, similar circuits might be used, requiring corresponding changes to be made in the software.

This program has been written with the assumption that a terminal will be connected to the UART. As such, only ASCII data can be transferred and each block of data is delineated by a carriage return (0DH) and line feed (0AH). As data is received by the UART it is stored in one of four rotating buffers. This data will later be transmitted by the GSC to other C152s. Data received by the GSC is stored in one of four different rotating buffers. This data will be transmitted by the UART to a terminal. 1K of external data RAM is connected to the C152 to serve as storage buffers. Consequently, each buffer is one-eighth of available external RAM, or 128 bytes. This provides up to one line of 120 characters for each buffer. Also, each buffer will store additional information such as destination address, source address, and message length. When a line of characters is complete, a flag will be set to signify to the GSC that that buffer is to be transmitted. Conversely, when a packet received by the GSC is complete, a flag is set to identify that buffer is to be output through the UART to a terminal. Whenever access to one buffer is complete, the software manipulates pointers so the next buffer is used. If all 4 buffers are full, data for that type of buffer is no longer accepted until another buffer is available.

Note that this program uses both DMA channels, one for the receiver and one for the transmitter on the GSC. A program could have been written using only one DMA channel. Using both channels has made the program much simpler and shortened the time it takes to change from transmitting to receiving.

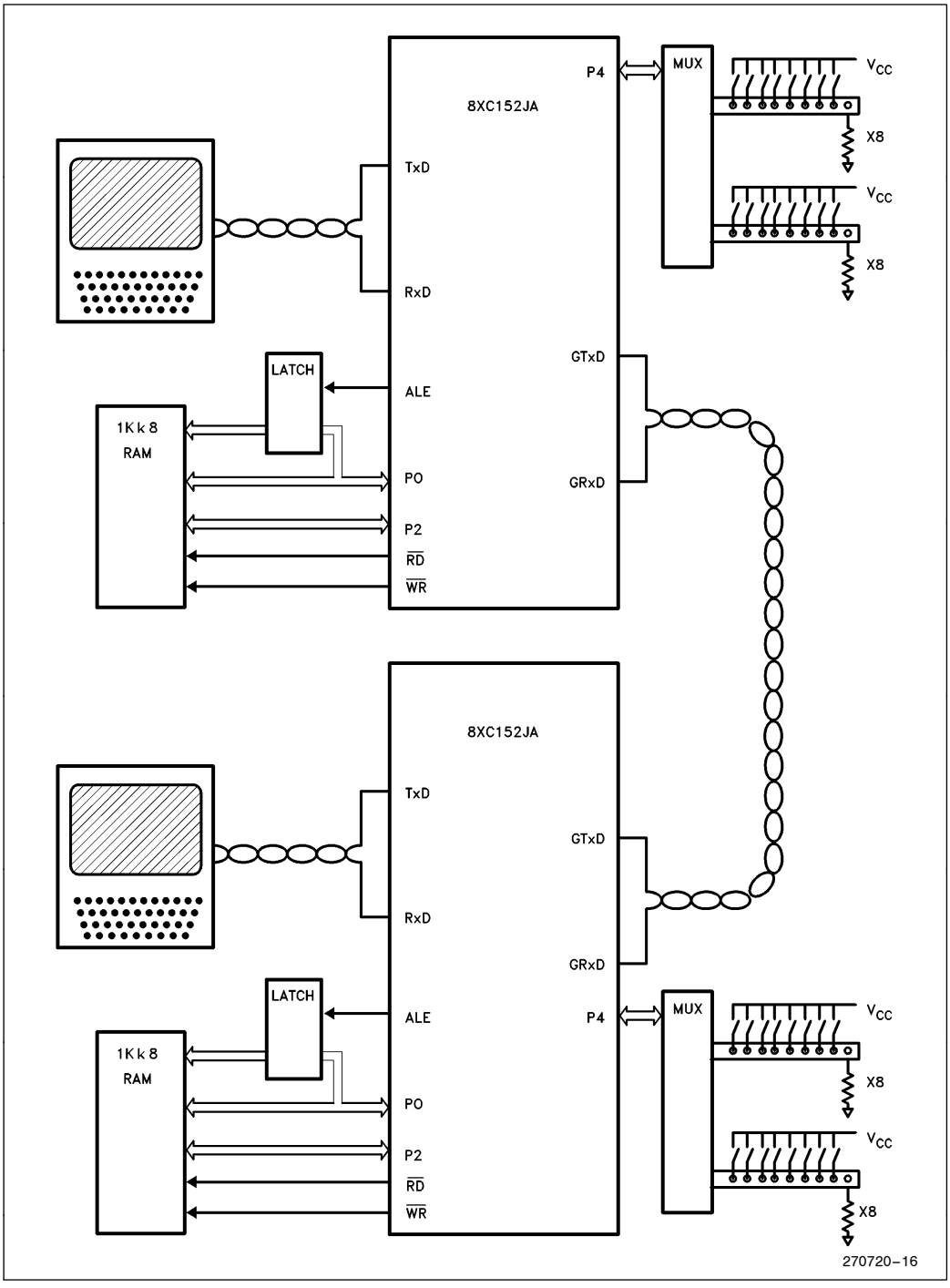

Figure 11. Hardware Environment for Software Example

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                    10/19/88   PAGE    1


DOS 3.30 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN APPNOT1.OBJ
ASSEMBLER INVOKED BY:  C:\ASM51\ASM51.EXE APPNOT1.PGM


LOC  OBJ            LINE     SOURCE

                       1     $XREF
                       2     $NOLIST
                     165
  0000               166     GSC_BAUD_RATE    EQU     0              ;GSC baud rate = 1.5MBPs
                     167
  00FC               168     LSC_BAUD_RATE    EQU     0FCH           ;LSC baud rate = 9.6K baud at
                     169                                             ;14.7456 MHz
                     170
  0014               171     IFS_PERIOD       EQU     20             ;number of bit times separating
                     172                                             ;frames
                     173
  0003               174     BUF1A_STRT_ADDR  EQU     003H           ;buffer 1A's starting address for
                     175                                             ;storing data (0 = # of bytes,
                     176                                             ;1 = dest addr, 2 = src addr)
                     177
  0083               178     BUF1B_STRT_ADDR  EQU     083H           ;buffer 1B's starting address for
                     179                                             ;storing data (80H = # of bytes,
                     180                                             ;81 = dest addr, 82 = src addr)
                     181
  0103               182     BUF1C_STRT_ADDR  EQU     103H           ;buffer 1C's starting address for
                     183                                             ;storing data (100H = # of bytes,
                     184                                             ;101 = dest addr, 102 = src addr)
                     185
  0183               186     BUF1D_STRT_ADDR  EQU     183H           ;buffer 1D's starting address for
                     187                                             ;storing data (180H = # of bytes,
                     188                                             ;181 = dest addr, 182 = src addr)
                     189
  0201               190     BUF2A_STRT_ADDR  EQU     201H           ;buffer 2A's starting address for
                     191                                             ;storing data (200H = # of bytes)
                     192
  0281               193     BUF2B_STRT_ADDR  EQU     281H           ;buffer 2B's starting address for
                     194                                             ;storing data (280H = # of bytes)
                     195
  0301               196     BUF2C_STRT_ADDR  EQU     301H           ;buffer 2C's starting address for
                     197                                             ;storing data (300H = # of bytes)
                     198
  0381               199     BUF2D_STRT_ADDR  EQU     381H           ;buffer 2D's starting address for
                     200                                             ;storing data (380H = # of bytes)
                     201
  0080               202     STACK_OFFSET     EQU     80H            ;start stack at upper 128 bytes
                     203
  000D               204     CR               EQU     0DH            ;ASCII equivalent for carriage
                     205                                             ;return
                     206
  000A               207     LINE_FEED        EQU     0AH            ;ASCII equivalent for line-feed
                     208
  REG                209     ERROR_POINTER    EQU     R0             ;R0 holds the address that points
                     210                                             ;to the next error location to
                     211                                             ;increment
                     212
```

270720-17

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                    10/19/88   PAGE    2

LOC  OBJ            LINE     SOURCE

  007B                213     MAX_LENGTH      EQU     120             ;maximum length a (received) packet
                      214                                             ;can be - must always be less than
                      215                                             ;255, my H/W limitation is 128
                      216
                      217     ;*******************************************************************************
                      218
  00FF                219     UR_COUNTER      DATA    0FFH            ;RAM locations 0FAH to 0FFH are
                      220                                             ;used to keep a log of the # of
                      221                                             ;UR errors (only transmit error)
                      222
  00F9                223     OVR_COUNTER     DATA    (UR_COUNTER) - 6    ;RAM locations 0F4H to 0F9H keep
                      224                                             ;a log of the # of overrun errors
                      225
  00F3                226     RCABT_COUNTER   DATA    (OVR_COUNTER) - 6   ;RAM locations 0EEH to 0F3H keep
                      227                                             ;a log of the # of abort errors
                      228
  00ED                229     AE_COUNTER      DATA    (RCABT_COUNTER) - 6 ;RAM locations 0E8H to 0EDH keep
                      230                                             ;a log of the # of alignment errors
                      231
  00E7                232     CRCE_COUNTER    DATA    (AE_COUNTER) - 6    ;RAM locations 0E2H to 0E7H keep
                      233                                             ;a log of the # of CRC errors
                      234
  00E1                235     LONG_COUNTER    DATA    (CRCE_COUNTER) - 6  ;RAM locations 0E1H to 0DCH keep
                      236                                             ;a log of the # of received
                      237                                             ;packets that are too long
                      238
  00DB                239     TCDT_COUNTER    DATA    (LONG_COUNTER) - 6  ;RAM locations 0DBH to 0D6H keep
                      240                                             ;a log of the # of TCDT errors
                      241
  00D5                242     NOACK_COUNTER   DATA    (TCDT_COUNTER) - 6  ;RAM locations 0D5H to 0D0H keep
                      243                                             ;a log of the # of NOACK errors
                      244
  00CF                245     NEXT_LOCATION   DATA    (NOACK_COUNTER) - 6 ;reserve 6 bytes for NOACK counter
                      246
                      247     ;-------------------------------------------------------------------------------
                      248
  007F                249     IN_BYTE_COUNT   DATA    7FH             ;number of bytes LSC received which
                      250                                             ;determines # of bytes for GSC to
                      251                                             ;transmit
                      252
  007E                253     OUT_BYTE_COUNT  DATA    (IN_BYTE_COUNT) -1  ;number of bytes GSC received which
                      254                                             ;determines # of bytes for LSC to
                      255                                             ;transmit
                      256
  007D                257     GSC_DEST_ADDR   DATA    (OUT_BYTE_COUNT) -1 ;destination address read from
                      258                                             ;DIP switches (loaded on RESET)
                      259
  007C                260     GSC_SRC_ADDR    DATA    (GSC_DEST_ADDR) -1  ;source address read from DIP
                      261                                             ;switches (loaded on RESET)
                      262
  007B                263     LSC_INPUT_LOW   DATA    (GSC_SRC_ADDR) -1
  007A                264     LSC_INPUT_HIGH  DATA    (LSC_INPUT_LOW) - 1 ;contains the address where the
                      265                                             ;next LSC received byte will be
                      266                                             ;stored at
                      267
```

270720-18

```
LOC  OBJ            LINE     SOURCE

  0079                268     GSC_INPUT_LOW   DATA    (LSC_INPUT_HIGH) - 1
  0078                269     GSC_INPUT_HIGH  DATA    (GSC_INPUT_LOW) - 1 ;contains the address where the
                      270                                                 ;next GSC received byte will be
                      271                                                 ;stored at
                      272
  0077                273     LSC_OUTPUT_LOW  DATA    (GSC_INPUT_HIGH) -1
  0076                274     LSC_OUTPUT_HIGH DATA    (LSC_OUTPUT_LOW) -1 ;contains the address where the
                      275                                                 ;next byte for the LSC to xmit
                      276
  0075                277     LSC_OUT_COUNTER DATA    (LSC_OUTPUT_HIGH)-1 ;contains the number of byte for
                      278                                                 ;the LSC to xmit
                      279
  002F                280     BUFFER1_CONTROL DATA    2FH                 ;byte that buffer 1 control bits
                      281                                                 ;are in
                      282
  002E                283     BUFFER2_CONTROL DATA    2EH                 ;byte that buffer 2 control bits
                      284                                                 ;are in
                      285
                      286     ;*************************************************************************
                      287
  007F                288     BUF1D_ACTIVE    BIT 7FH                     ;indicator for when buffer 1D has
                      289                                                 ;data for GSC
                      290
  007E                291     BUF1C_ACTIVE    BIT (BUF1D_ACTIVE) - 1      ;indicator for when buffer 1C has
                      292                                                 ;data for GSC
                      293
  007D                294     BUF1B_ACTIVE    BIT (BUF1C_ACTIVE) - 1      ;indicator for when buffer 1B has
                      295                                                 ;data for GSC
                      296
  007C                297     BUF1A_ACTIVE    BIT (BUF1B_ACTIVE) - 1      ;indicator for when buffer 1A has
                      298                                                 ;data for GSC
                      299
  007B                300     GSC_OUT_MSB     BIT (BUF1A_ACTIVE) - 1      ;second of two bits that identify
                      301                                                 ;which buffer is the current GSC
                      302                                                 ;output buffer
                      303
  007A                304     GSC_OUT_LSB     BIT (GSC_OUT_MSB) - 1       ;first of two bits that identify
                      305                                                 ;which buffer is the current GSC
                      306                                                 ;output buffer
                      307
  0079                308     LSC_IN_MSB      BIT (GSC_OUT_LSB) - 1       ;second of two bits that identify
                      309                                                 ;which buffer is the current LSC
                      310                                                 ;input buffer
                      311
  0078                312     LSC_IN_LSB      BIT (LSC_IN_MSB) - 1        ;first of two bits that identify
                      313                                                 ;which buffer is the current LSC
                      314                                                 ;input buffer
                      315
  0077                316     BUF2A_ACTIVE    BIT (LSC_IN_LSB) - 1        ;indicator for when buffer 2A has
                      317                                                 ;data for LSC
                      318
  0076                319     BUF2B_ACTIVE    BIT (BUF2A_ACTIVE) - 1      ;indicator for when buffer 2B has
                      320                                                 ;data for LSC
                      321
  0075                322     BUF2C_ACTIVE    BIT (BUF2B_ACTIVE) - 1      ;indicator for when buffer 2C has
```

270720-19

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                                10/19/88   PAGE    4

LOC  OBJ            LINE     SOURCE

                     323                                          ;data for LSC
                     324
  0074               325     BUF2D_ACTIVE    BIT (BUF2C_ACTIVE) - 1       ;indicator for when buffer 2D has
                     326                                          ;data for LSC
                     327
  0073               328     GSC_IN_MSB      BIT (BUF2D_ACTIVE) - 1       ;second of two bits that identify
                     329                                          ;which buffer is the current GSC
                     330                                          ;input buffer
                     331
  0072               332     GSC_IN_LSB      BIT (GSC_IN_MSB) - 1         ;first of two bits that identify
                     333                                          ;which buffer is the current GSC
                     334                                          ;input buffer
                     335
  0071               336     LSC_OUT_MSB     BIT (GSC_IN_LSB) - 1         ;second of two bits that identify
                     337                                          ;which buffer is the current LSC
                     338                                          ;output buffer
                     339
  0070               340     LSC_OUT_LSB     BIT (LSC_OUT_MSB) - 1        ;first of two bits that identify
                     341                                          ;which buffer is the current LSC
                     342                                          ;output buffer
                     343
  006F               344     FIRST_GSC_OUT   BIT (LSC_OUT_LSB) - 1        ;indicator for first GSC xmit
                     345
  006E               346     LSC_ACTIVE      BIT (FIRST_GSC_OUT) -1       ;indicator that LSC is outputting
                     347                                          ;a received packet
                     348
                     349     ;*****************************************************************************
                     350
                     351     START:
0000                 352     ORG 0
0000 020100          353     JMP INITIALIZATION
                     354
0023                 355     ORG 23H
0023 0205BA          356     JMP LSC_SERVICE
                     357
002B                 358     ORG 2BH
                     359     GSC_REC_VALID:
002B 020568          360         JMP GSC_VALID_REC
                     361
0033                 362     ORG 33H
                     363     GSC_REC_ERROR:
0033 020580          364         JMP GSC_ERROR_REC
                     365
0043                 366     ORG 43H
                     367     GSC_XMIT_VALID:
0043 0204AA          368         JMP GSC_VALID_XMIT
                     369
004B                 370     ORG 4BH
                     371     GSC_XMIT_ERROR:
004B 0204E3          372         JMP GSC_ERROR_XMIT
                     373
0053                 374     ORG 53H
                     375     DMA1_DONE:
0053 02061C          376         JMP DMA1_SERVICE
                     377
```

270720-20

```
LOC  OBJ             LINE     SOURCE

0100                  378      ORG 100H
                      379      INITIALIZATION:
                      380
0100 758180           381          MOV SP,#STACK_OFFSET              ;start stack at user defined addr
                      382
0103 120243           383          CALL ADDRESS_DETERMINATION        ;setup addressing (only done on
                      384                                            ;RESET)
                      385
0106 120200           386          CALL GSC_INIT                     ;initialization for GSC
                      387
0109 120234           388          CALL LSC_INIT                     ;initialization for LSC
                      389
010C 12025B           390          CALL GENERIC_INIT                 ;general initialization not dealing
                      391                                            ;with interrupts, GSC, or LSC
                      392
010F 120250           393          CALL INTERRUPT_ENABLE             ;enable interrupts
                      394
                      395      MAIN:
                      396
0112 207C17           397          JB BUF1A_ACTIVE,BUFFER1_START     ;see if buffer 1A has something
                      398                                            ;to transmit out GSC
                      399
0115 207D14           400          JB BUF1B_ACTIVE,BUFFER1_START     ;see if buffer 1B has something
                      401                                            ;to transmit out GSC
                      402
0118 207E11           403          JB BUF1C_ACTIVE,BUFFER1_START     ;see if buffer 1C has something
                      404                                            ;to transmit out GSC
                      405
011B 207F0E           406          JB BUF1D_ACTIVE,BUFFER1_START     ;see if buffer 1D has something
                      407                                            ;to transmit out GSC
                      408
011E 207710           409          JB BUF2A_ACTIVE,BUFFER2_START     ;see if buffer 2A has something
                      410                                            ;to transmit out LSC
                      411
0121 20760D           412          JB BUF2B_ACTIVE,BUFFER2_START     ;see if buffer 2B has something
                      413                                            ;to transmit out LSC
                      414
0124 20750A           415          JB BUF2C_ACTIVE,BUFFER2_START     ;see if buffer 2C has something
                      416                                            ;to transmit out LSC
                      417
0127 207407           418          JB BUF2D_ACTIVE,BUFFER2_START     ;see if buffer 2D has something
                      419                                            ;to transmit out LSC
                      420
012A 80E6             421          JMP MAIN
                      422
                      423      BUFFER1_START:
                      424
012C 12032F           425          CALL NEW_BUFFER1_OUT              ;this routine should start a
                      426                                            ;transmission if a buffer is full
                      427
012F 80E1             428          JMP MAIN
                      429
                      430      BUFFER2_START:
                      431
0131 12043F           432          CALL NEW_BUFFER2_OUT              ;this routine starts a transmission
```

270720-21

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                    10/19/88   PAGE    6

LOC  OBJ            LINE     SOURCE

                     433                                          ;out the LSC if one of the buffers
                     434                                          ;is full
                     435
0134 80DC            436          JMP MAIN
                     437
0200                 438     ORG 200H
                     439 +1  $INCLUDE (GSCINIT.SRC)
                =1   440     GSC_INIT:
                =1   441
0200 759400     =1   442          MOV BAUD,#GSC_BAUD_RATE
                =1   443
0203 758402     =1   444          MOV GMOD,#02H                   ;init for CSMA/CD, 8-bit preamble,
                =1   445                                          ;16-bit CRC, 8-bit addresses
                =1   446
0206 75A414     =1   447          MOV IFS,#IFS_PERIOD             ;init IFS for period between frames
                =1   448
0209 75D400     =1   449          MOV TCDCNT,#0                   ;clear collision counter
                =1   450
020C D2D8       =1   451          SETB DMA                        ;init GSC interrupts for DMA
                =1   452
020E 75C285     =1   453          MOV DARL0,#TFIFO                ;DMA0 will service TFIFO
                =1   454
0211 759298     =1   455          MOV DCON0,#10011000B            ;init DMA0 with SFR as dest, ext RAM
                =1   456                                          ;as source, serial port demand mode
                =1   457
0214 75B2F4     =1   458          MOV SARL1,#RFIFO                ;DMA1 will service RFIFO
                =1   459
0217 75F300     =1   460          MOV BCRH1,#0
021A 75F278     =1   461          MOV BCRL1,#MAX_LENGTH           ;load DMA byte count with maximum
                =1   462                                          ;message length
                =1   463
021D 759369     =1   464          MOV DCON1,#01101001B            ;init DMA1 with ext RAM as dest,
                =1   465                                          ;SFR as source, serial port demand
                =1   466                                          ;mode, and set GO bit.
                =1   467
                =1   468
0220 857C95     =1   469          MOV ADR0,GSC_SRC_ADDR
                =1   470
0223 757901     =1   471          MOV GSC_INPUT_LOW,#LOW (BUF2A_STRT_ADDR)
0226 757802     =1   472          MOV GSC_INPUT_HIGH,#HIGH (BUF2A_STRT_ADDR)
                =1   473                                          ;init GSC input address storage
                =1   474
0229 8579D2     =1   475          MOV DARL1,GSC_INPUT_LOW
022C 8578D3     =1   476          MOV DARH1,GSC_INPUT_HIGH        ;init DMA destination address to
                =1   477                                          ;match GSC input address storage
                =1   478
022F D2E9       =1   479          SETB GREN                       ;enable receiver
                =1   480
0231 D26F       =1   481          SETB FIRST_GSC_OUT              ;set indicator that first GSC xmit
                =1   482                                          ;has not yet occurred
                =1   483
0233 22         =1   484          RET
                     485 +1  $INCLUDE (LSCINIT.SRC)
                =1   486     LSC_INIT:
0234 758DFC     =1   487          MOV TH1,#LSC_BAUD_RATE          ;setup timer1 to generate LSC baud
```

270720-22

```
LOC  OBJ            LINE     SOURCE

                    =1   488
0237 438920         =1   489         ORL TMOD,#00100000B
023A 53892F         =1   490         ANL TMOD,#00101111B            ;init timer1 as 8-bit auto-reload
                    =1   491
023D 759850         =1   492         MOV SCON,#01010000B            ;setup LSC as 8-bit UART and enable
                    =1   493                                        ;receiver
                    =1   494
0240 D28E           =1   495         SETB TR1                       ;start timer to generate baud rate
                    =1   496
0242 22             =1   497         RET
                         498 +1  $INCLUDE (INITADDR.SRC)
                    =1   499     ADDRESS_DETERMINATION:
                    =1   500
0243 53901F         =1   501         ANL P1,#1FH                        ;select output 0 of '138
                    =1   502
0246 85C07C         =1   503         MOV GSC_SRC_ADDR,P4                ;read GSC receive address from
                    =1   504                                            ;DIP switch #1
                    =1   505
0249 439020         =1   506         ORL P1,#20H                        ;select output 1 of '138
                    =1   507
024C 85C07D         =1   508         MOV GSC_DEST_ADDR,P4               ;read GSC xmit address from DIP
                    =1   509                                            ;switch #2
                    =1   510
024F 22             =1   511         RET
                         512 +1  $INCLUDE (ENAINT.SRC)
                    =1   513     INTERRUPT_ENABLE:
                    =1   514
0250 D2C8           =1   515         SETB EGSRV                     ;enable GSC receive valid interrupt
                    =1   516
0252 D2C9           =1   517         SETB EGSRE                     ;enable GSC receive error interrupt
                    =1   518
0254 D2AC           =1   519         SETB ES                        ;enable LSC interrupt
                    =1   520
0256 D2CC           =1   521         SETB EDMA1                     ;enable DMA1 done interrupt
                    =1   522
0258 D2AF           =1   523         SETB EA                        ;enable interrupts
                    =1   524
025A 22             =1   525         RET
                    =1   526
                         527 +1  $INCLUDE (GENINIT.SRC)
                    =1   528     GENERIC_INIT:
                    =1   529
025B 752F00         =1   530         MOV BUFFER1_CONTROL,#0                 ;insure all buffer 1 active bits
                    =1   531                                                ;= 0, current input and output
                    =1   532                                                ;buffer = 1A
                    =1   533
                    =1   534
025E 752E00         =1   535         MOV BUFFER2_CONTROL,#0                 ;insure all buffer 2 active bits
                    =1   536                                                ;= 0, current input and output
                    =1   537                                                ;buffer = 1B
                    =1   538
0261 C26E           =1   539         CLR LSC_ACTIVE                         ;insure LSC_ACTIVE = 0 before
                    =1   540                                                ;starting a reception
                    =1   541
0263 757B03         =1   542         MOV LSC_INPUT_LOW,#LOW (BUF1A_STRT_ADDR)
```

270720-23

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                          10/19/88   PAGE    8

LOC  OBJ            LINE     SOURCE

0266 757A00      =1    543         MOV LSC_INPUT_HIGH,#HIGH (BUF1A_STRT_ADDR)
                 =1    544                                              ;load address pointers with
                 =1    545                                              ;starting address of buffer 1A
                 =1    546
0269 757F02      =1    547         MOV IN_BYTE_COUNT,#02                ;byte count initialized to 2
                 =1    548                                              ;because destination and source
                 =1    549                                              ;address will take first two bytes
                 =1    550                                              ;and counter is not incremented.
                 =1    551
026C 78CF        =1    552         MOV R0,#NEXT_LOCATION
                 =1    553
                 =1    554     COUNTER_CLEAR:
026E 08          =1    555         INC R0
                 =1    556
026F 7600        =1    557         MOV @R0,#0                       ;clear out error counter area
                 =1    558
0271 B8FFFA      =1    559         CJNE R0,#0FFH,COUNTER_CLEAR      ;loop until all counters = 0
                 =1    560
0274 22          =1    561         RET
                 =1    562
                 =1    563
                       564 +1  $INCLUDE (CNTRINC.SRC)
                 =1    565     INCREMENT_COUNTER:
                 =1    566
0275 D3          =1    567         SETB C                           ;add 1 on first loop
                 =1    568
0276 7F06        =1    569         MOV R7,#6                        ;# of bytes in each counter field
                 =1    570
                 =1    571     INC_COUNT_LOOP:
                 =1    572
0278 E6          =1    573         MOV A,@ERROR_POINTER             ;get byte of counter
                 =1    574
0279 3400        =1    575         ADDC A,#0
                 =1    576
027B F6          =1    577         MOV @ERROR_POINTER,A
                 =1    578
027C 18          =1    579         DEC ERROR_POINTER
                 =1    580
027D DFF9        =1    581         DJNZ R7,INC_COUNT_LOOP
                 =1    582
027F 4001        =1    583         JC COUNTER_OVERFLOW              ;overflow if carry generated.  This
                 =1    584                                          ;was initially put in to stop the
                 =1    585                                          ;flow of the program if any of the
                 =1    586                                          ;error counters overflowed with the
                 =1    587                                          ;expectation that the user would
                 =1    588                                          ;modify the code to dump the error
                 =1    589                                          ;count contents and re-initialize the
                 =1    590                                          ;counter locations.
                 =1    591
0281 22          =1    592         RET
                 =1    593
                 =1    594     COUNTER_OVERFLOW:
                 =1    595
0282 08          =1    596         INC ERROR_POINTER                ;point to msb of counter field
                 =1    597
```

270720-24

```
LOC  OBJ            LINE     SOURCE

0283 76FF       =1   598         MOV @ERROR_POINTER,#0FFH        ;and store OFFH
                =1   599
0285 08         =1   600         INC ERROR_POINTER               ;point to next byte of coutner field
                =1   601
0286 76FF       =1   602         MOV @ERROR_POINTER,#0FFH        ;and store OFFH
                =1   603
0288 08         =1   604         INC ERROR_POINTER               ;point to next byte of counter field
                =1   605
0289 76FF       =1   606         MOV @ERROR_POINTER,#0FFH        ;and store OFFH
                =1   607
028B 08         =1   608         INC ERROR_POINTER               ;point to next byte of counter field
                =1   609
028C 76FF       =1   610         MOV @ERROR_POINTER,#0FFH        ;and store OFFH
                =1   611
028E 08         =1   612         INC ERROR_POINTER               ;point to next byte of counter field
                =1   613
028F 76FF       =1   614         MOV @ERROR_POINTER,#0FFH        ;and store OFFH
                =1   615
0291 08         =1   616         INC ERROR_POINTER               ;point to next byte of counter field
                =1   617
0292 76FF       =1   618         MOV @ERROR_POINTER,#0FFH        ;and store OFFH
                =1   619
0294 80FE       =1   620         JMP $                           ;if the error counters overflow the
                =1   621                                         ;program continues to loop at this
                =1   622                                         ;location until H/W resets the device.
                     623 +1  $INCLUDE (BUF1MGT.SRC)
                =1   624     NEW_BUFFER1_IN:
                =1   625
                =1   626     ;*******************************************************************************
                =1   627     ;This section uses a bit addressable control byte to determine which buffers
                =1   628     ;are active (contains data for GSC to output), the last buffer used by the LSC
                =1   629     ;input, and the last buffer used by the GSC output.
                =1   630     ;
                =1   631     ;The control byte is defined as follows:
                =1   632     ;
                =1   633     ;                                                  00 = BUFFER 1A
                =1   634     ;                                                  01 = BUFFER 1B
                =1   635     ;                                                  10 = BUFFER 1C
                =1   636     ;                                                  11 = BUFFER 1D
                =1   637     ;                                                 ____¦____
                =1   638     ;                                                 ¦       ¦
                =1   639     ;                                        LAST BUFFER USED    LAST BUFFER USED
                =1   640     ;                                        BY GSC FOR OUTPUT   BY LSC FOR INPUT
                =1   641     ;                                           ___¦___             ___¦___
                =1   642     ;                                          ¦       ¦           ¦       ¦
                =1   643     ;  ____________________________________¦_______¦_______¦_______¦_______¦____
                =1   644     ;  ¦ BIT 7 ¦ BIT 6 ¦ BIT 5 ¦ BIT 4 ¦ BIT 3 ¦ BIT 2 ¦ BIT 1 ¦ BIT 0 ¦
                =1   645     ;  ¦_______¦_______¦_______¦_______¦_______¦_______¦_______¦_______¦
                =1   646     ;      ¦       ¦       ¦       ¦       ¦       ¦       ¦       ¦
                =1   647     ;      ¦ BUFFER 1C   ¦ BUFFER 1A   ¦       ¦       ¦       ¦
                =1   648     ;      ¦   ACTIVE    ¦   ACTIVE    ¦       ¦       ¦       ¦
                =1   649     ;  BUFFER 1D   ¦ BUFFER 1B   ¦ GSC_OUT_MSB  ¦ LSC_IN_MSB   ¦
                =1   650     ;    ACTIVE    ¦   ACTIVE    ¦             ¦             ¦
                =1   651     ;      ¦ BUF1C_ACT   ¦ BUF1A_ACT     GSC_OUT_LSB     LSC_IN_MSB
                =1   652     ;      ¦             ¦
```

270720-25

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                          10/19/88   PAGE   10

LOC  OBJ            LINE     SOURCE

                    =1   653     ;    BUF1D_ACT        BUF1B_ACT
                    =1   654     ;
                    =1   655     ;*************************************************************************
                    =1   656
0296 20794E         =1   657         JB LSC_IN_MSB,LSC_IN_1D_1A              ;if LSC_IN_MSB = 1 (1C or 1D),
                    =1   658                                                 ;then the next buffer to be used
                    =1   659                                                 ;must be 1D or 1A.
                    =1   660
0299 207823         =1   661         JB LSC_IN_LSB,LSC_IN_1C                 ;if LSC_IN = 01B then next buffer
                    =1   662                                                 ;for LSC to use is 1C
                    =1   663
                    =1   664     LSC_IN_1B                                   ;if LSC_IN = 00B (only combination
                    =1   665                                                 ;left) then next buffer to use is
                    =1   666                                                 ;1B
                    =1   667
029C 207D43         =1   668         JB BUF1B_ACTIVE,BUFFERS_1_FULL          ;if buffer 1B is active then the
                    =1   669                                                 ;GSC has not yet emptied it and
                    =1   670                                                 ;all the buffers must be full
                    =1   671
029F 758200         =1   672         MOV DPL,#LOW (BUF1A_STRT_ADDR) - 3
02A2 758300         =1   673         MOV DPH,#HIGH (BUF1A_STRT_ADDR)         ;setup DPTR to point at the
                    =1   674                                                 ;beginning of buffer 1A (first byte
                    =1   675                                                 ;should contain number of bytes
                    =1   676
02A5 E57F           =1   677         MOV A,IN_BYTE_COUNT                     ;load acc with byte count for MOVX
                    =1   678
02A7 F0             =1   679         MOVX @DPTR,A                            ;store byte count at first byte of
                    =1   680                                                 ;buffer 1A
                    =1   681
02A8 A3             =1   682         INC DPTR                                ;DPTR now points to where the
                    =1   683                                                 ;destination address should be
                    =1   684
02A9 E57D           =1   685         MOV A,GSC_DEST_ADDR                     ;get stored destination address
                    =1   686
02AB F0             =1   687         MOVX @DPTR,A                            ;store destination addr in XRAM
                    =1   688
02AC A3             =1   689         INC DPTR                                ;DPTR now points to where source
                    =1   690                                                 ;address should be stored
                    =1   691
02AD E57C           =1   692         MOV A,GSC_SRC_ADDR                      ;get stored source address
                    =1   693
02AF F0             =1   694         MOVX @DPTR,A                            ;store destination addr in XRAM
                    =1   695
02B0 D27C           =1   696         SETB BUF1A_ACTIVE                       ;indicate that BUF1A has data to
                    =1   697                                                 ;be output by the GSC and that the
                    =1   698                                                 ;LSC has moved on to the next
                    =1   699                                                 ;buffer
                    =1   700
02B2 C279           =1   701         CLR LSC_IN_MSB                          ;set flags to indicate that the
02B4 D278           =1   702         SETB LSC_IN_LSB                         ;current input buffer (for LSC)
                    =1   703                                                 ;is 1B
                    =1   704
02B6 757B83         =1   705         MOV LSC_INPUT_LOW,#LOW (BUF1B_STRT_ADDR)
02B9 757A00         =1   706         MOV LSC_INPUT_HIGH,#HIGH (BUF1B_STRT_ADDR)
                    =1   707                                                 ;load starting address of buffer
```

270720-26

```
LOC  OBJ            LINE     SOURCE

                =1   708                                        ;1B
                =1   709
02BC 02032D     =1   710        JMP NEW_BUF1_IN_END
                =1   711
                =1   712
                =1   713    LSC_IN_1C:
                =1   714
02BF 207E20     =1   715        JB BUF1C_ACTIVE,BUFFERS_1_FULL  ;if buffer 1C is active then the
                =1   716                                        ;GSC has not yet emptied it and
                =1   717                                        ;all the buffers must be full
                =1   718
02C2 758280     =1   719        MOV DPL,#LOW (BUF1B_STRT_ADDR) - 3
02C5 758300     =1   720        MOV DPH,#HIGH (BUF1B_STRT_ADDR) ;setup DPTR to point at the
                =1   721                                        ;beginning of buffer 1B (first byte
                =1   722                                        ;should contain number of bytes
                =1   723
02C8 E57F       =1   724        MOV A,IN_BYTE_COUNT             ;load acc with byte count for MOVX
                =1   725
02CA F0         =1   726        MOVX @DPTR,A                    ;store byte count at first byte of
                =1   727                                        ;buffer 1B
                =1   728
02CB A3         =1   729        INC DPTR                        ;DPTR now points to where the
                =1   730                                        ;destination address should be
                =1   731
02CC E57D       =1   732        MOV A,GSC_DEST_ADDR             ;get stored destination address
                =1   733
02CE F0         =1   734        MOVX @DPTR,A                    ;store destination addr in XRAM
                =1   735
02CF A3         =1   736        INC DPTR                        ;DPTR now points to where source
                =1   737                                        ;address should be stored
                =1   738
02D0 E57C       =1   739        MOV A,GSC_SRC_ADDR              ;get stored source address
                =1   740
02D2 F0         =1   741        MOVX @DPTR,A                    ;store destination addr in XRAM
                =1   742
02D3 D27D       =1   743        SETB BUF1B_ACTIVE               ;indicate that BUF1C has data to
                =1   744                                        ;be output by the GSC and that the
                =1   745                                        ;LSC has moved on to the next
                =1   746                                        ;buffer
                =1   747
02D5 C278       =1   748        CLR LSC_IN_LSB                  ;set flags to indicate that the
02D7 D279       =1   749        SETB LSC_IN_MSB                 ;current input buffer (for LSC)
                =1   750                                        ;is 1C
                =1   751
02D9 757B03     =1   752        MOV LSC_INPUT_LOW,#LOW (BUF1C_STRT_ADDR)
02DC 757A01     =1   753        MOV LSC_INPUT_HIGH,#HIGH (BUF1C_STRT_ADDR)
                =1   754                                        ;load starting address of buffer
                =1   755                                        ;1C
                =1   756
02DF 02032D     =1   757        JMP NEW_BUF1_IN_END
                =1   758
                =1   759    BUFFERS_1_FULL:
                =1   760
02E2 12032E     =1   761        CALL IRET                       ;if the buffers are full, the pgm
                =1   762                                        ;will be locked in the LSC service
```

270720-27

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                    10/19/88   PAGE   12

LOC  OBJ            LINE     SOURCE

                    =1   763                                        ;routine in an "interrupt in
                    =1   764                                        ;progress" mode.  If the DMA then
                    =1   765                                        ;frees up a buffer, the interrupt
                    =1   766                                        ;routine cannot clear the buffer
                    =1   767                                        ;active bit until the interrupt
                    =1   768                                        ;(EGSTV/EGSTE) is serviced
                    =1   769
02E5 80AF           =1   770        JMP NEW_BUFFER1_IN              ;continue scanning active buffers
                    =1   771                                        ;until one is freed up
                    =1   772
                    =1   773    LSC_IN_1D_1A:
                    =1   774
02E7 207823         =1   775        JB LSC_IN_LSB,LSC_IN_1A         ;if LSC_IN = 11 then next buffer
                    =1   776                                        ;next buffer is 1A
                    =1   777
                    =1   778    LSC_IN_1D:
                    =1   779
02EA 207FF5         =1   780        JB BUF1D_ACTIVE,BUFFERS_1_FULL  ;if buffer 1D is active then the
                    =1   781                                        ;GSC has not yet emptied it and
                    =1   782                                        ;all the buffers must be full
                    =1   783
02ED 758200         =1   784        MOV DPL,#LOW (BUF1C_STRT_ADDR) - 3
02F0 758301         =1   785        MOV DPH,#HIGH (BUF1C_STRT_ADDR) ;setup DPTR to point at the
                    =1   786                                        ;beginning of buffer 1C (first byte
                    =1   787                                        ;should contain number of bytes
                    =1   788
02F3 E57F           =1   789        MOV A,IN_BYTE_COUNT             ;load acc with byte count for MOVX
                    =1   790
02F5 F0             =1   791        MOVX @DPTR,A                    ;store byte count at first byte of
                    =1   792                                        ;buffer 1C
                    =1   793
02F6 A3             =1   794        INC DPTR                        ;DPTR now points to where the
                    =1   795                                        ;destination address should be
                    =1   796
02F7 E57D           =1   797        MOV A,GSC_DEST_ADDR             ;get stored destination address
                    =1   798
02F9 F0             =1   799        MOVX @DPTR,A                    ;store destination addr in XRAM
                    =1   800
02FA A3             =1   801        INC DPTR                        ;DPTR now points to where source
                    =1   802                                        ;address should be stored
                    =1   803
02FB E57C           =1   804        MOV A,GSC_SRC_ADDR              ;get stored source address
                    =1   805
02FD F0             =1   806        MOVX @DPTR,A                    ;store destination addr in XRAM
                    =1   807
02FE D27E           =1   808        SETB BUF1C_ACTIVE               ;indicate that BUF1C has data to
                    =1   809                                        ;be output by the GSC and that the
                    =1   810                                        ;LSC has moved on to the next
                    =1   811                                        ;buffer
                    =1   812
0300 D278           =1   813        SETB LSC_IN_LSB                 ;set flags to indicate that the
0302 D279           =1   814        SETB LSC_IN_MSB                 ;current input buffer (for LSC)
                    =1   815                                        ;is 1D
                    =1   816
0304 757B83         =1   817        MOV LSC_INPUT_LOW,#LOW (BUF1D_STRT_ADDR)
```

270720-28

```
LOC  OBJ            LINE     SOURCE

0307 757A01      =1   818         MOV LSC_INPUT_HIGH,#HIGH (BUF1D_STRT_ADDR)
                 =1   819                                              ;load starting address of buffer
                 =1   820                                              ;1D
                 =1   821
030A 02032D      =1   822         JMP NEW_BUF1_IN_END
                 =1   823
                 =1   824     LSC_IN_1A:
                 =1   825
030D 207CD2      =1   826         JB BUF1A_ACTIVE,BUFFERS_1_FULL        ;if buffer 1A is active then the
                 =1   827                                              ;GSC has not yet emptied it and
                 =1   828                                              ;all the buffers must be full
                 =1   829
0310 758280      =1   830         MOV DPL,#LOW (BUF1D_STRT_ADDR)   ;3
0313 758301      =1   831         MOV DPH,#HIGH (BUF1D_STRT_ADDR)       ;setup DPTR to point at the
                 =1   832                                              ;beginning of buffer 1D (first byte
                 =1   833                                              ;should contain number of bytes
                 =1   834
0316 E57F        =1   835         MOV A,IN_BYTE_COUNT                   ;load acc with byte count for MOVX
                 =1   836
0318 F0          =1   837         MOVX @DPTR,A                          ;store byte count at first byte of
                 =1   838                                              ;buffer 1A
                 =1   839
0319 A3          =1   840         INC DPTR                              ;DPTR now points to where the
                 =1   841                                              ;destination address should be
                 =1   842
031A E57D        =1   843         MOV A,GSC_DEST_ADDR                   ;get stored destination address
                 =1   844
031C F0          =1   845         MOVX @DPTR,A                          ;store destination addr in XRAM
                 =1   846
031D A3          =1   847         INC DPTR                              ;DPTR now points to where source
                 =1   848                                              ;address should be stored
                 =1   849
031E E57C        =1   850         MOV A,GSC_SRC_ADDR                    ;get stored source address
                 =1   851
0320 F0          =1   852         MOVX @DPTR,A                          ;store destination addr in XRAM
                 =1   853
0321 D27F        =1   854         SETB BUF1D_ACTIVE                     ;indicate that BUF1D has data to
                 =1   855                                              ;be output by the GSC and that the
                 =1   856                                              ;LSC has moved on to the next
                 =1   857                                              ;buffer
                 =1   858
0323 C278        =1   859         CLR LSC_IN_LSB                        ;set flags to indicate that the
0325 C279        =1   860         CLR LSC_IN_MSB                        ;current input buffer (for LSC)
                 =1   861                                              ;is 1A
                 =1   862
0327 757B03      =1   863         MOV LSC_INPUT_LOW,#LOW (BUF1A_STRT_ADDR)
032A 757A00      =1   864         MOV LSC_INPUT_HIGH,#HIGH (BUF1A_STRT_ADDR)
                 =1   865                                              ;load starting address of buffer
                 =1   866                                              ;1A
                 =1   867
                 =1   868     NEW_BUF1_IN_END:
032D 22          =1   869         RET
                 =1   870
                 =1   871
                 =1   872     IRET:
```

270720-29

```
MCS-51 MACRO ASSEMBLER   APPNOT1                                  10/19/88  PAGE   14

LOC  OBJ            LINE     SOURCE

032E 32          =1   873         RETI                          ;re-enable interrupts
                 =1   874
                 =1   875     NEW_BUFFER1_OUT:
                 =1   876
032F 30D903      =1   877         JNB TEN,SECOND_TEN_CHECK      ;do not start another transmission
                 =1   878                                       ;if one is in progress (signified
                 =1   879                                       ;by TEN = 1) but this should never
                 =1   880                                       ;happen
                 =1   881     TRANSMISSION_IN_PROGRESS:
0332 0203AF      =1   882         JMP NOTHING_FOR_GSC           ;do not start a new GSC xmit if one
                 =1   883                                       ;is currently in progress
                 =1   884
                 =1   885     SECOND_TEN_CHECK:
0335 20D9FA      =1   886         JB TEN,TRANSMISSION_IN_PROGRESS ;second one in case interrupt
                 =1   887                                       ;occurs during previous test
                 =1   888
0338 207B37      =1   889         JB GSC_OUT_MSB,GSC_OUT_1C_1D  ;if GSC_OUT_MSB = 1 then current
                 =1   890                                       ;buffer is 1C or 1D
                 =1   891
033B 207A1A      =1   892         JB GSC_OUT_LSB,GSC_OUT_1B     ;if GSC_OUT = 01B then current
                 =1   893                                       ;buffer is 1B
                 =1   894
                 =1   895     GSC_OUT_1A:                       ;if GSC_OUT = 00B then the buffer
                 =1   896                                       ;is 1A
                 =1   897
033E 307C6E      =1   898         JNB BUF1A_ACTIVE,NOTHING_FOR_GSC ;if buffer 1A is not active then
                 =1   899                                       ;the LSC has not yet filled it
                 =1   900                                       ;since the GSC emptied it last
                 =1   901
0341 900000      =1   902         MOV DPTR,#(BUF1A_STRT_ADDR) -3 ;load DPTR with address of byte
                 =1   903                                       ;that holds byte count for 1A
                 =1   904
0344 E0          =1   905         MOVX A,@DPTR                  ;get byte count for buffer 1A
                 =1   906
0345 F5E2        =1   907         MOV BCRLO,A                   ;load DMA byte count with length
                 =1   908                                       ;of message to transmit
                 =1   909
0347 75E300      =1   910         MOV BCRHO,#0                  ;insure high byte count = 0
                 =1   911                                       ;(should already be 0)
                 =1   912
034A A3          =1   913         INC DPTR                      ;DPTR now points at dest addr
                 =1   914
034B 8582A2      =1   915         MOV SARLO,DPL
034E 8583A3      =1   916         MOV SARHO,DPH                 ;source address for start of
                 =1   917                                       ;data to send
                 =1   918
0351 C27B        =1   919         CLR GSC_OUT_MSB
0353 D27A        =1   920         SETB GSC_OUT_LSB              ;indicate next output buffer will
                 =1   921                                       ;be buffer 1B
                 =1   922
0355 0203A6      =1   923         JMP START_GSC_OUT             ;routine that starts transmission
                 =1   924
                 =1   925     GSC_OUT_1B:                       ;if GSC_OUT = 01B then the buffer
                 =1   926                                       ;is 1B
                 =1   927
```

270720-30

```
LOC  OBJ            LINE     SOURCE

0358 307D54     =1   928         JNB BUF1B_ACTIVE,NOTHING_FOR_GSC        ;if buffer 1B is not active then
                =1   929                                                 ;the LSC has not yet filled it
                =1   930                                                 ;since the GSC emptied it last
                =1   931
035B 900080     =1   932         MOV DPTR,#(BUF1B_STRT_ADDR) -3          ;load DPTR with address of byte
                =1   933                                                 ;that holds byte count for 1B
                =1   934
035E E0         =1   935         MOVX A,@DPTR                            ;get byte count for buffer 1B
                =1   936
035F F5E2       =1   937         MOV BCRLO,A                             ;load DMA byte count with length
                =1   938                                                 ;of message to transmit
                =1   939
0361 75E300     =1   940         MOV BCRHO,#0                            ;insure high byte count = 0
                =1   941                                                 ;(should already be 0)
                =1   942
0364 A3         =1   943         INC DPTR                                ;DPTR now points at dest addr
                =1   944
0365 8582A2     =1   945         MOV SARLO,DPL
0368 8583A3     =1   946         MOV SARHO,DPH                           ;source address for start of
                =1   947                                                 ;data to send
                =1   948
036B D27B       =1   949         SETB GSC_OUT_MSB
036D C27A       =1   950         CLR GSC_OUT_LSB                         ;indicate next output buffer will
                =1   951                                                 ;be buffer 1C
                =1   952
036F 0203A6     =1   953         JMP START_GSC_OUT                       ;routine that starts transmission
                =1   954
                =1   955     GSC_OUT_1C_1D:
                =1   956
0372 207A1A     =1   957         JB GSC_OUT_LSB,GSC_OUT_1D               ;output buffer will be 1D if
                =1   958                                                 ;GSC_OUT = 11B
                =1   959
                =1   960
                =1   961     GSC_OUT_1C:                                 ;if GSC_OUT = 10B then the buffer
                =1   962                                                 ;is 1C
                =1   963
0375 307E37     =1   964         JNB BUF1C_ACTIVE,NOTHING_FOR_GSC        ;if buffer 1C is not active then
                =1   965                                                 ;the LSC has not yet filled it
                =1   966                                                 ;since the GSC emptied it last
                =1   967
0378 900100     =1   968         MOV DPTR,#(BUF1C_STRT_ADDR) -3          ;load DPTR with address of byte
                =1   969                                                 ;that holds byte count for 1C
                =1   970
037B E0         =1   971         MOVX A,@DPTR                            ;get byte count for buffer 1C
                =1   972
037C F5E2       =1   973         MOV BCRLO,A                             ;load DMA byte count with length
                =1   974                                                 ;of message to transmit
                =1   975
037E 75E300     =1   976         MOV BCRHO,#0                            ;insure high byte count = 0
                =1   977                                                 ;(should already be 0)
                =1   978
0381 A3         =1   979         INC DPTR                                ;DPTR now points at dest addr
                =1   980
0382 8582A2     =1   981         MOV SARLO,DPL
0385 8583A3     =1   982         MOV SARHO,DPH                           ;source address for start of
```

270720-31

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                    10/19/88   PAGE   16

LOC  OBJ            LINE     SOURCE

                    =1   983                                    ;data to send
                    =1   984
0388 D27B           =1   985        SETB GSC_OUT_MSB
038A D27A           =1   986        SETB GSC_OUT_LSB            ;indicate next output buffer will
                    =1   987                                    ;be buffer 1D
                    =1   988
038C 0203A6         =1   989        JMP START_GSC_OUT           ;routine that starts transmission
                    =1   990
                    =1   991    GSC_OUT_1D:                     ;if GSC_OUT = 11B then the buffer
                    =1   992                                    ;is 1D
                    =1   993
038F 307F1D         =1   994        JNB BUF1D_ACTIVE,NOTHING_FOR_GSC ;if buffer 1D is not active then
                    =1   995                                    ;the LSC has not yet filled it
                    =1   996                                    ;since the GSC emptied it last
                    =1   997
0392 900180         =1   998        MOV DPTR,#(BUF1D_STRT_ADDR) -3 ;load DPTR with address of byte
                    =1   999                                    ;that holds byte count for 1D
                    =1  1000
0395 E0             =1  1001        MOVX A,@DPTR                ;get byte count for buffer 1D
                    =1  1002
0396 F5E2           =1  1003        MOV BCRLO,A                 ;load DMA byte count with length
                    =1  1004                                    ;of message to transmit
                    =1  1005
0398 75E300         =1  1006        MOV BCRHO,#0                ;insure high byte count = 0
                    =1  1007                                    ;(should already be 0)
                    =1  1008
039B A3             =1  1009        INC DPTR                    ;DPTR now points at dest addr
                    =1  1010
039C 8582A2         =1  1011        MOV SARLO,DPL
039F 8583A3         =1  1012        MOV SARHO,DPH               ;source address for start of
                    =1  1013                                    ;data to send
                    =1  1014
03A2 C27B           =1  1015        CLR GSC_OUT_MSB
03A4 C27A           =1  1016        CLR GSC_OUT_LSB             ;indicate next output buffer will
                    =1  1017                                    ;be buffer 1A
                    =1  1018
                    =1  1019    START_GSC_OUT:                  ;routine that starts transmission
                    =1  1020
03A6 D2D9           =1  1021        SETB TEN                    ;enable GSC transmitter
                    =1  1022
03A8 D2CB           =1  1023        SETB EGSTV                  ;enable GSC transmit valid (TDN)
                    =1  1024                                    ;interrupt
                    =1  1025
03AA D2CD           =1  1026        SETB EGSTE                  ;enable GSC transmit error int
                    =1  1027
03AC 439201         =1  1028        ORL DCONO,#01               ;start DMA which starts data output
                    =1  1029
                    =1  1030    NOTHING_FOR_GSC:
                    =1  1031
03AF 22             =1  1032        RET
                    =1  1033
                    =1  1034
                        1035 +1 $INCLUDE (BUF2MGT.SRC)
                    =1  1036    NEW_BUFFER2_IN:
                    =1  1037
```

270720-32

```
LOC  OBJ            LINE     SOURCE

                    =1  1038 ;*******************************************************************************
                    =1  1039 ;This section uses a bit addressable control byte to determine which buffers
                    =1  1040 ;are active (contains data for LSC to output), the last buffer used by the LSC
                    =1  1041 ;for output, and the last buffer used by the GSC for input
                    =1  1042 ;
                    =1  1043 ;The control byte is defined as follows:
                    =1  1044 ;
                    =1  1045 ;                                                  00 = BUFFER 2A
                    =1  1046 ;                                                  01 = BUFFER 2B
                    =1  1047 ;                                                  10 = BUFFER 2C
                    =1  1048 ;                                                  11 = BUFFER 2D
                    =1  1049 ;                                                 _____¦_____
                    =1  1050 ;                                                 ¦     ¦
                    =1  1051 ;                                        LAST BUFFER USED    LAST BUFFER USED
                    =1  1052 ;                                        BY GSC FOR INPUT    BY LSC FOR OUTPUT
                    =1  1053 ;                                                ___¦___          ___¦___
                    =1  1054 ;                                               ¦       ¦        ¦       ¦
                    =1  1055 ;  _____________________________________________¦_______¦_______¦_______¦_____
                    =1  1056 ;  ¦ BIT 7 ¦ BIT 6 ¦ BIT 5 ¦ BIT 4 ¦ BIT 3 ¦ BIT 2 ¦ BIT 1 ¦ BIT 0 ¦
                    =1  1057 ;  ¦_______¦_______¦_______¦_______¦_______¦_______¦_______¦_______¦
                    =1  1058 ;      ¦       ¦       ¦       ¦       ¦       ¦       ¦       ¦
                    =1  1059 ;      ¦  BUFFER 2C  ¦  BUFFER 2A  ¦       ¦       ¦       ¦
                    =1  1060 ;      ¦   ACTIVE    ¦   ACTIVE    ¦       ¦       ¦       ¦
                    =1  1061 ;  BUFFER 2D   ¦  BUFFER 2B  ¦  GSC_IN_MSB  ¦ LSC_OUT_MSB  ¦
                    =1  1062 ;   ACTIVE     ¦   ACTIVE    ¦             ¦             ¦
                    =1  1063 ;      ¦  BUF2C_ACT  ¦  BUF2A_ACT      GSC_IN_LSB    LSC_OUT_MSB
                    =1  1064 ;      ¦             ¦
                    =1  1065 ;  BUF2D_ACT      BUF2B_ACT
                    =1  1066 ;
                    =1  1067 ;*******************************************************************************
                    =1  1068
03B0 207343         =1  1069     JB GSC_IN_MSB,GSC_IN_2D_2A          ;if GSC_IN_MSB = 1 (2C or 2D),
                    =1  1070                                         ;then the next buffer to be used
                    =1  1071                                         ;must be 2D or 2A.
                    =1  1072
03B3 20721E         =1  1073     JB GSC_IN_LSB,GSC_IN_2C             ;if GSC_IN = 01B then next buffer
                    =1  1074                                         ;for GSC to use is 2C
                    =1  1075
                    =1  1076 GSC_IN_2B:                              ;if GSC_IN = 00B (only combination
                    =1  1077                                         ;left) then next buffer to use is
                    =1  1078                                         ;2B
                    =1  1079
03B6 207639         =1  1080     JB BUF2B_ACTIVE,BUFFERS_2_FULL      ;if buffer 2B is active then the
                    =1  1081                                         ;LSC has not yet emptied it and
                    =1  1082                                         ;all the buffers must be full
                    =1  1083
03B9 758200         =1  1084     MOV DPL,#LOW (BUF2A_STRT_ADDR) - 1
03BC 758302         =1  1085     MOV DPH,#HIGH (BUF2A_STRT_ADDR)     ;setup DPTR to point at the
                    =1  1086                                         ;beginning of buffer 2A (first byte
                    =1  1087                                         ;should contain number of bytes
                    =1  1088
03BF C3             =1  1089     CLR C                               ;for SUBB
                    =1  1090
03C0 7476           =1  1091     MOV A,#(MAX_LENGTH) - 2             ;maximum packet length and the
                    =1  1092                                         ;initial value for BCRL1 (-2
```

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                    10/19/88   PAGE   18

LOC  OBJ            LINE     SOURCE

                    =1  1093                                        ;subtracted because first 2 bytes
                    =1  1094                                        ;are the destination and source
                    =1  1095                                        ;addresses
                    =1  1096
03C2 95F2           =1  1097        SUBB A,BCRL1                    ;load acc with byte count for MOVX
                    =1  1098
03C4 F0             =1  1099        MOVX @DPTR,A                    ;store byte count at first byte of
                    =1  1100                                        ;buffer 2A
                    =1  1101
03C5 D277           =1  1102        SETB BUF2A_ACTIVE               ;indicate that BUF2A has data to
                    =1  1103                                        ;be output by the LSC and that the
                    =1  1104                                        ;GSC has moved on to the next
                    =1  1105                                        ;buffer
                    =1  1106
03C7 C273           =1  1107        CLR GSC_IN_MSB                  ;set flags to indicate that the
03C9 D272           =1  1108        SETB GSC_IN_LSB                 ;current input buffer (for GSC)
                    =1  1109                                        ;is 2B
                    =1  1110
03CB 757981         =1  1111        MOV GSC_INPUT_LOW,#LOW (BUF2B_STRT_ADDR)
03CE 757802         =1  1112        MOV GSC_INPUT_HIGH,#HIGH (BUF2B_STRT_ADDR)
                    =1  1113                                        ;load starting address of buffer
                    =1  1114                                        ;2B
                    =1  1115
03D1 020432         =1  1116        JMP NEW_BUF2_IN_END
                    =1  1117
                    =1  1118
                    =1  1119    GSC_IN_2C:
                    =1  1120
03D4 20751B         =1  1121        JB BUF2C_ACTIVE,BUFFERS_2_FULL  ;if buffer 2C is active then the
                    =1  1122                                        ;LSC has not yet emptied it and
                    =1  1123                                        ;all the buffers must be full
                    =1  1124
03D7 758280         =1  1125        MOV DPL,#LOW (BUF2B_STRT_ADDR) - 1
03DA 758302         =1  1126        MOV DPH,#HIGH (BUF2B_STRT_ADDR) ;setup DPTR to point at the
                    =1  1127                                        ;beginning of buffer 2B (first byte
                    =1  1128                                        ;should contain number of bytes
                    =1  1129
03DD C3             =1  1130        CLR C                           ;for SUBB
                    =1  1131
03DE 7476           =1  1132        MOV A,#(MAX_LENGTH) - 2         ;maximum packet length and the
                    =1  1133                                        ;initial value for BCRL1 ( 2
                    =1  1134                                        ;subtracted because first 2 bytes
                    =1  1135                                        ;are the destination and source
                    =1  1136                                        ;addresses
                    =1  1137
03E0 95F2           =1  1138        SUBB A,BCRL1                    ;load acc with byte count for MOVX
                    =1  1139
03E2 F0             =1  1140        MOVX @DPTR,A                    ;store byte count at first byte of
                    =1  1141                                        ;buffer 2B
                    =1  1142
03E3 D276           =1  1143        SETB BUF2B_ACTIVE               ;indicate that BUF2B has data to
                    =1  1144                                        ;be output by the LSC and that the
                    =1  1145                                        ;GSC has moved on to the next
                    =1  1146                                        ;buffer
                    =1  1147
```

270720-34

```
LOC  OBJ             LINE     SOURCE

03E5 C272        =1  1148         CLR GSC_IN_LSB                      ;set flags to indicate that the
03E7 D273        =1  1149         SETB GSC_IN_MSB                     ;current input buffer (for GSC)
                 =1  1150                                             ;is 2C
                 =1  1151
03E9 757901      =1  1152         MOV GSC_INPUT_LOW,#LOW (BUF2C_STRT_ADDR)
03EC 757803      =1  1153         MOV GSC_INPUT_HIGH,#HIGH (BUF2C_STRT_ADDR)
                 =1  1154                                             ;load starting address of buffer
                 =1  1155                                             ;2C
                 =1  1156
03EF 020432      =1  1157         JMP NEW_BUF2_IN_END
                 =1  1158
                 =1  1159     BUFFERS_2_FULL:
                 =1  1160
03F2 712E        =1  1161         CALL IRET                           ;if the buffers are full, the pgm
                 =1  1162                                             ;will be locked in the GSC service
                 =1  1163                                             ;routine in an "interrupt in
                 =1  1164                                             ;progress" mode.  If the DMA then
                 =1  1165                                             ;frees up a buffer, the interrupt
                 =1  1166                                             ;routine cannot clear the buffer
                 =1  1167                                             ;active bit until the interrupt
                 =1  1168                                             ;(EGSRV/EGSRE) is serviced
                 =1  1169
03F4 80BA        =1  1170         JMP NEW_BUFFER2_IN                  ;continue scanning active buffers
                 =1  1171                                             ;until one is freed up
                 =1  1172
                 =1  1173     GSC_IN_2D_2A:
                 =1  1174
03F6 20721E      =1  1175         JB GSC_IN_LSB,GSC_IN_2A             ;if GSC_IN = 11 then next buffer
                 =1  1176                                             ;next buffer is 2A
                 =1  1177
                 =1  1178     GSC_IN_2D:
                 =1  1179
03F9 2074F6      =1  1180         JB BUF2D_ACTIVE,BUFFERS_2_FULL      ;if buffer 2D is active then the
                 =1  1181                                             ;LSC has not yet emptied it and
                 =1  1182                                             ;all the buffers must be full
                 =1  1183
03FC 758200      =1  1184         MOV DPL,#LOW (BUF2C_STRT_ADDR) - 1
03FF 758303      =1  1185         MOV DPH,#HIGH (BUF2C_STRT_ADDR)     ;setup DPTR to point at the
                 =1  1186                                             ;beginning of buffer 2C (first byte
                 =1  1187                                             ;should contain number of bytes
                 =1  1188
0402 C3          =1  1189         CLR C                               ;for SUBB
                 =1  1190
0403 7476        =1  1191         MOV A,#(MAX_LENGTH) - 2             ;maximum packet length and the
                 =1  1192                                             ;initial value for BCRL1 ( 2
                 =1  1193                                             ;subtracted because first 2 bytes
                 =1  1194                                             ;are the destination and source
                 =1  1195                                             ;addresses
                 =1  1196
0405 95F2        =1  1197         SUBB A,BCRL1                        ;load acc with byte count for MOVX
                 =1  1198
0407 F0          =1  1199         MOVX @DPTR,A                        ;store byte count at first byte of
                 =1  1200                                             ;buffer 2C
                 =1  1201
0408 D275        =1  1202         SETB BUF2C_ACTIVE                   ;indicate that BUF2C has data to
```

270720-35

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                    10/19/88   PAGE   20

LOC  OBJ            LINE     SOURCE

                    =1  1203                                        ;be output by the LSC and that the
                    =1  1204                                        ;GSC has moved on to the next
                    =1  1205                                        ;buffer
                    =1  1206
040A D272           =1  1207        SETB GSC_IN_LSB                 ;set flags to indicate that the
040C D273           =1  1208        SETB GSC_IN_MSB                 ;current input buffer (for GSC)
                    =1  1209                                        ;is 2D
                    =1  1210
040E 757981         =1  1211        MOV GSC_INPUT_LOW,#LOW (BUF2D_STRT_ADDR)
0411 757803         =1  1212        MOV GSC_INPUT_HIGH,#HIGH (BUF2D_STRT_ADDR)
                    =1  1213                                        ;load starting address of buffer
                    =1  1214                                        ;2D
                    =1  1215
0414 020432         =1  1216        JMP NEW_BUF2_IN_END
                    =1  1217
                    =1  1218    GSC_IN_2A:
                    =1  1219
0417 2077D8         =1  1220        JB BUF2A_ACTIVE,BUFFERS_2_FULL  ;if buffer 2A is active then the
                    =1  1221                                        ;LSC has not yet emptied it and
                    =1  1222                                        ;all the buffers must be full
                    =1  1223
041A 758280         =1  1224        MOV DPL,#LOW (BUF2D_STRT_ADDR) - 1
041D 758303         =1  1225        MOV DPH,#HIGH (BUF2D_STRT_ADDR) ;setup DPTR to point at the
                    =1  1226                                        ;beginning of buffer 2D (first byte
                    =1  1227                                        ;should contain number of bytes
                    =1  1228
0420 C3             =1  1229        CLR C                           ;for SUBB
                    =1  1230
0421 7476           =1  1231        MOV A,#(MAX_LENGTH) - 2         ;maximum packet length and the
                    =1  1232                                        ;initial value for BCRL1 ( 2
                    =1  1233                                        ;subtracted because first 2 bytes
                    =1  1234                                        ;are the destination and source
                    =1  1235                                        ;addresses
                    =1  1236
0423 95F2           =1  1237        SUBB A,BCRL1                    ;load acc with byte count for MOVX
                    =1  1238
0425 F0             =1  1239        MOVX @DPTR,A                    ;store byte count at first byte of
                    =1  1240                                        ;buffer 2A
                    =1  1241
0426 D274           =1  1242        SETB BUF2D_ACTIVE               ;indicate that BUF2D has data to
                    =1  1243                                        ;be output by the LSC and that the
                    =1  1244                                        ;GSC has moved on to the next
                    =1  1245                                        ;buffer
                    =1  1246
0428 C272           =1  1247        CLR GSC_IN_LSB                  ;set flags to indicate that the
042A C273           =1  1248        CLR GSC_IN_MSB                  ;current input buffer (for GSC)
                    =1  1249                                        ;is 2A
                    =1  1250
042C 757901         =1  1251        MOV GSC_INPUT_LOW,#LOW (BUF2A_STRT_ADDR)
042F 757802         =1  1252        MOV GSC_INPUT_HIGH,#HIGH (BUF2A_STRT_ADDR)
                    =1  1253                                        ;load starting address of buffer
                    =1  1254                                        ;2A
                    =1  1255
                    =1  1256    NEW_BUF2_IN_END:
                    =1  1257
```

270720-36

```
LOC  OBJ             LINE     SOURCE

0432 8579D2      =1  1258         MOV DARL1,GSC_INPUT_LOW
0435 8578D3      =1  1259         MOV DARH1,GSC_INPUT_HIGH          ;load DMA destination address
                 =1  1260                                           ;registers with starting address
                 =1  1261                                           ;of current buffer area
                 =1  1262
0438 75F300      =1  1263         MOV BCRH1,#0
043B 75F278      =1  1264         MOV BCRL1,#MAX_LENGTH             ;load DMA byte count with packet
                 =1  1265                                           ;length
043E 22          =1  1266         RET
                 =1  1267
                 =1  1268
                 =1  1269     NEW_BUFFER2_OUT:
                 =1  1270
043F 306E03      =1  1271         JNB LSC_ACTIVE,SECOND_LSC_CHECK   ;do not start another transmission
                 =1  1272                                           ;if one is in progress (signified
                 =1  1273                                           ;by LSC_ACTIVE = 1) but this
                 =1  1274                                           ;should never happen
                 =1  1275
                 =1  1276     LSC_XMIT_IN_PROGRESS:
0442 0204A9      =1  1277         JMP NOTHING_FOR_LSC               ;do not start a new LSC xmit if one
                 =1  1278                                           ;is currently in progress
                 =1  1279
                 =1  1280     SECOND_LSC_CHECK:
0445 206EFA      =1  1281         JB LSC_ACTIVE,LSC_XMIT_IN_PROGRESS ;second one in case interrupt
                 =1  1282                                           ;occurs during previous test
                 =1  1283
0448 20712B      =1  1284         JB LSC_OUT_MSB,LSC_OUT_2C_2D      ;if LSC_OUT_MSB = 1 then current
                 =1  1285                                           ;buffer is 2C or 2D
                 =1  1286
044B 207014      =1  1287         JB LSC_OUT_LSB,LSC_OUT_2B         ;if LSC_OUT = 01B then current
                 =1  1288                                           ;buffer is 2B
                 =1  1289
                 =1  1290     LSC_OUT_2A:                           ;if LSC_OUT = 00B then the buffer
                 =1  1291                                           ;is 2A
                 =1  1292
044E 307758      =1  1293         JNB BUF2A_ACTIVE,NOTHING_FOR_LSC  ;if buffer 2A is not active then
                 =1  1294                                           ;the GSC has not yet filled it
                 =1  1295                                           ;since the LSC emptied it last
                 =1  1296
0451 D26E        =1  1297         SETB LSC_ACTIVE                   ;show that LSC is in the process of
                 =1  1298                                           ;doing a transmission
                 =1  1299
0453 900200      =1  1300         MOV DPTR,#(BUF2A_STRT_ADDR) -1    ;load DPTR with address of byte
                 =1  1301                                           ;that holds byte count for 2A
                 =1  1302
0456 E0          =1  1303         MOVX A,@DPTR                      ;get byte count for buffer 2A
                 =1  1304
0457 F575        =1  1305         MOV LSC_OUT_COUNTER,A             ;load LSC byte counter with length
                 =1  1306                                           ;of message to transmit
                 =1  1307
0459 0575        =1  1308         INC LSC_OUT_COUNTER               ;incremented because the counter
                 =1  1309                                           ;is first decremented before being
                 =1  1310                                           ;tested (DJNZ) when LSC begins to
                 =1  1311                                           ;output data
                 =1  1312
```

270720-37

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                    10/19/88   PAGE   22

LOC  OBJ            LINE     SOURCE

045B C271          =1  1313         CLR LSC_OUT_MSB
045D D270          =1  1314         SETB LSC_OUT_LSB                ;indicate next output buffer will
                   =1  1315                                         ;be buffer 2B
                   =1  1316
045F 02049E        =1  1317         JMP START_LSC_OUT               ;routine that starts transmission
                   =1  1318
                   =1  1319     LSC_OUT_2B:                         ;if LSC_OUT = 01B then the buffer
                   =1  1320                                         ;is 2B
                   =1  1321
0462 307644        =1  1322         JNB BUF2B_ACTIVE,NOTHING_FOR_LSC ;if buffer 2B is not active then
                   =1  1323                                         ;the GSC has not yet filled it
                   =1  1324                                         ;since the LSC emptied it last
                   =1  1325
0465 D26E          =1  1326         SETB LSC_ACTIVE                 ;show that LSC is in the process of
                   =1  1327                                         ;doing a transmission
                   =1  1328
0467 900280        =1  1329         MOV DPTR,#(BUF2B_STRT_ADDR) -1  ;load DPTR with address of byte
                   =1  1330                                         ;that holds byte count for 2B
                   =1  1331
046A E0            =1  1332         MOVX A,@DPTR                    ;get byte count for buffer 2B
                   =1  1333
046B F575          =1  1334         MOV LSC_OUT_COUNTER,A           ;load LSC byte counter with length
                   =1  1335                                         ;of message to transmit
                   =1  1336
046D 0575          =1  1337         INC LSC_OUT_COUNTER             ;incremented because the counter
                   =1  1338                                         ;is first decremented before being
                   =1  1339                                         ;tested (DJNZ) when LSC begins to
                   =1  1340                                         ;output data
                   =1  1341
046F D271          =1  1342         SETB LSC_OUT_MSB
0471 C270          =1  1343         CLR LSC_OUT_LSB                 ;indicate next output buffer will
                   =1  1344                                         ;be buffer 2C
                   =1  1345
0473 02049E        =1  1346         JMP START_LSC_OUT               ;routine that starts transmission
                   =1  1347
                   =1  1348     LSC_OUT_2C_2D:
                   =1  1349
0476 207014        =1  1350         JB LSC_OUT_LSB,LSC_OUT_2D       ;if LSC_OUT = 11B then current
                   =1  1351                                         ;buffer is 2D
                   =1  1352
                   =1  1353
                   =1  1354     LSC_OUT_2C:                         ;if LSC_OUT = 10B then the buffer
                   =1  1355                                         ;is 2C
                   =1  1356
0479 30752D        =1  1357         JNB BUF2C_ACTIVE,NOTHING_FOR_LSC ;if buffer 2C is not active then
                   =1  1358                                         ;the GSC has not yet filled it
                   =1  1359                                         ;since the LSC emptied it last
                   =1  1360
047C D26E          =1  1361         SETB LSC_ACTIVE                 ;show that LSC is in the process of
                   =1  1362                                         ;doing a transmission
                   =1  1363
047E 900300        =1  1364         MOV DPTR,#(BUF2C_STRT_ADDR) -1  ;load DPTR with address of byte
                   =1  1365                                         ;that holds byte count for 2C
                   =1  1366
0481 E0            =1  1367         MOVX A,@DPTR                    ;get byte count for buffer 2C~
```

270720-38

```
LOC  OBJ            LINE     SOURCE

                    =1  1368
0482 F575           =1  1369         MOV LSC_OUT_COUNTER,A              ;load LSC byte counter with length
                    =1  1370                                            ;of message to transmit
                    =1  1371
0484 0575           =1  1372         INC LSC_OUT_COUNTER                ;incremented because the counter
                    =1  1373                                            ;is first decremented before being
                    =1  1374                                            ;tested (DJNZ) when LSC begins to
                    =1  1375                                            ;output data
                    =1  1376
0486 D271           =1  1377         SETB LSC_OUT_MSB
0488 D270           =1  1378         SETB LSC_OUT_LSB                   ;indicate next output buffer will
                    =1  1379                                            ;be buffer 2D
                    =1  1380
048A 02049E         =1  1381         JMP START_LSC_OUT                  ;routine that starts transmission
                    =1  1382
                    =1  1383     LSC_OUT_2D:                            ;if LSC_OUT = 11B then the buffer
                    =1  1384                                            ;is 2D
                    =1  1385
048D 307419         =1  1386         JNB BUF2D_ACTIVE,NOTHING_FOR_LSC   ;if buffer 2D is not active then
                    =1  1387                                            ;the GSC has not yet filled it
                    =1  1388                                            ;since the LSC emptied it last
                    =1  1389
0490 D26E           =1  1390         SETB LSC_ACTIVE                    ;show that LSC is in the process of
                    =1  1391                                            ;doing a transmission
                    =1  1392
0492 900380         =1  1393         MOV DPTR,#(BUF2D_STRT_ADDR) -1     ;load DPTR with address of byte
                    =1  1394                                            ;that holds byte count for 2D
                    =1  1395
0495 E0             =1  1396         MOVX A,@DPTR                       ;get byte count for buffer 2A
                    =1  1397
0496 F575           =1  1398         MOV LSC_OUT_COUNTER,A              ;load LSC byte counter with length
                    =1  1399                                            ;of message to transmit
                    =1  1400
0498 0575           =1  1401         INC LSC_OUT_COUNTER                ;incremented because the counter
                    =1  1402                                            ;is first decremented before being
                    =1  1403                                            ;tested (DJNZ) when LSC begins to
                    =1  1404                                            ;output data
                    =1  1405
049A C271           =1  1406         CLR LSC_OUT_MSB
049C C270           =1  1407         CLR LSC_OUT_LSB                    ;indicate next output buffer will
                    =1  1408                                            ;be buffer 2B
                    =1  1409
                    =1  1410     START_LSC_OUT:                         ;routine that starts transmission
                    =1  1411
049E A3             =1  1412         INC DPTR                           ;DPTR now points at the destination
                    =1  1413                                            ;address that was received
                    =1  1414
049F A3             =1  1415         INC DPTR                           ;DPTR now points at the source
                    =1  1416                                            ;address that was received
                    =1  1417
04A0 A3             =1  1418         INC DPTR                           ;DPTR now points at the first data
                    =1  1419                                            ;byte received
                    =1  1420
04A1 858277         =1  1421         MOV LSC_OUTPUT_LOW,DPL
04A4 858376         =1  1422         MOV LSC_OUTPUT_HIGH,DPH            ;address for start of data for LSC
```

270720-39

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                    10/19/88   PAGE   24

LOC  OBJ            LINE     SOURCE

                    =1  1423                                      ;to send
                    =1  1424
04A7 D299           =1  1425          SETB TI                     ;set interrupt flag to start
                    =1  1426                                      ;transmitting when main program is
                    =1  1427                                      ;returned to
                    =1  1428
                    =1  1429      NOTHING_FOR_LSC:
                    =1  1430
04A9 22             =1  1431          RET
                    =1  1432
                    =1  1433
                        1434 +1   $INCLUDE (XMITVAL.SRC)
                    =1  1435      GSC_VALID_XMIT:
                    =1  1436
04AA C082           =1  1437          PUSH DPL
04AC C083           =1  1438          PUSH DPH
04AE C0E0           =1  1439          PUSH ACC
04B0 C0D0           =1  1440          PUSH PSW                    ;SFRs to save before servicing
                    =1  1441                                      ;interrupt
                    =1  1442
                    =1  1443      ;*****************************************************************
                    =1  1444      ; DISABLE TRANSMIT INTERRUPTS
                    =1  1445      ;*****************************************************************
                    =1  1446
04B2 C2CB           =1  1447          CLR EGSTV                   ;clear valid interrupt enable
                    =1  1448
04B4 C2CD           =1  1449          CLR EGSTE                   ;clear error interrupt enable
                    =1  1450
                    =1  1451
                    =1  1452      CLEAR_ACTIVE_BUFFER:
                    =1  1453
04B6 207B12         =1  1454          JB GSC_OUT_MSB,CLEAR_ACTIVE_1B_1C
                    =1  1455                                      ;if GSC_OUT_MSB = 1 then
                    =1  1456                                      ;previous used buffer for GSC
                    =1  1457                                      ;output must have been 1B or 1C
                    =1  1458
04B9 207A08         =1  1459          JB GSC_OUT_LSB,CLEAR_ACTIVE_1A  ;if GSC_OUT = 01B then active
                    =1  1460                                      ;buffer 1A bit must be cleared
                    =1  1461
                    =1  1462      CLEAR_ACTIVE_1D:
                    =1  1463
04BC 206F16         =1  1464          JB FIRST_GSC_OUT,END_CLEAR_ACTIVE_OUT
                    =1  1465                                      ;if this is first transmission,
                    =1  1466                                      ;do not clear buffer 1D active
                    =1  1467                                      ;bit (this may happen if all
                    =1  1468                                      ;four buffers are filled before
                    =1  1469                                      ;first GSC transmission)
                    =1  1470
04BF C27F           =1  1471          CLR BUF1D_ACTIVE            ;if GSC_OUT = 00, then last
                    =1  1472                                      ;buffer used is 1D unless first
                    =1  1473                                      ;transmission
                    =1  1474
04C1 0204D5         =1  1475          JMP END_CLEAR_ACTIVE_OUT
                    =1  1476
                    =1  1477      CLEAR_ACTIVE_1A:
```

270720-40

```
LOC  OBJ            LINE     SOURCE

                    =1  1478
04C4 C27C           =1  1479         CLR BUF1A_ACTIVE                    ;if GSC_OUT = 01, then last
                    =1  1480                                             ;buffer used is 1A
                    =1  1481
04C6 C26F           =1  1482         CLR FIRST_GSC_OUT                   ;clear indicator that shows
                    =1  1483                                             ;the first GSC transmission
                    =1  1484                                             ;has not yet occurred
                    =1  1485
04C8 0204D5         =1  1486         JMP END_CLEAR_ACTIVE_OUT
                    =1  1487
                    =1  1488     CLEAR_ACTIVE_1B_1C:
                    =1  1489
04CB 207A05         =1  1490         JB GSC_OUT_LSB,CLEAR_ACTIVE_1C      ;if GSC_OUT = 11B, then last
                    =1  1491                                             ;buffer used is 1C
                    =1  1492
                    =1  1493     CLEAR_ACTIVE_1B:
                    =1  1494
04CE C27D           =1  1495         CLR BUF1B_ACTIVE                    ;if GSC_OUT = 10, then last
                    =1  1496                                             ;buffer used is 1B
                    =1  1497
04D0 0204D5         =1  1498         JMP END_CLEAR_ACTIVE_OUT
                    =1  1499
                    =1  1500     CLEAR_ACTIVE_1C:
                    =1  1501
04D3 C27E           =1  1502         CLR BUF1C_ACTIVE                    ;if GSC_OUT = 11, then last
                    =1  1503                                             ;buffer used is 1C unless
                    =1  1504                                             ;first transmission
                    =1  1505
                    =1  1506
                    =1  1507     END_CLEAR_ACTIVE_OUT:
                    =1  1508
                    =1  1509     ;*********************************************************************
                    =1  1510     ; SEE IF NEXT BUFFER IS FULL OR INIT ADDRESS FOR NEXT AVAIL BUFFER
                    =1  1511     ; WHEN IT IS FILLED
                    =1  1512     ;*********************************************************************
                    =1  1513
04D5 712F           =1  1514         CALL NEW_BUFFER1_OUT
                    =1  1515
                    =1  1516     ;*********************************************************************
                    =1  1517     ; RETURN TO MAIN PROGRAM LOOP
                    =1  1518     ;*********************************************************************
                    =1  1519
04D7 75D400         =1  1520         MOV TCDCNT,#0                       ;clear collision counter
                    =1  1521
04DA D0D0           =1  1522         POP PSW
04DC D0E0           =1  1523         POP ACC
04DE D083           =1  1524         POP DPH
04E0 D082           =1  1525         POP DPL                             ;SFRs that were saved
                    =1  1526
04E2 32             =1  1527         RETI
                    =1  1528
                        1529 +1  $INCLUDE (XMITERR.SRC)
                    =1  1530     GSC_ERROR_XMIT:
                    =1  1531
                    =1  1532     ;*********************************************************************
```

270720-41

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                    10/19/88   PAGE   26

LOC  OBJ            LINE     SOURCE

                    =1  1533     ; STOP DMA CHANNEL
                    =1  1534     ;*******************************************************************
                    =1  1535
04E3 5392FE         =1  1536         ANL DCONO,#0FEH                          ;clear GO bit
                    =1  1537
04E6 C082           =1  1538         PUSH DPL
04E8 C083           =1  1539         PUSH DPH
04EA C0E0           =1  1540         PUSH ACC
04EC C0D0           =1  1541         PUSH PSW                                 ;SFRs to save before servicing
                    =1  1542                                                  ;interrupt
                    =1  1543
                    =1  1544     UR_ERROR:
                    =1  1545
04EE 30DD05         =1  1546         JNB UR,NOACK_ERROR                       ;see if error caused by
                    =1  1547                                                  ;underrun
                    =1  1548
04F1 78FF           =1  1549         MOV ERROR_POINTER,#UR_COUNTER            ;load pointer with beginning
                    =1  1550                                                  ;address of UR counter
                    =1  1551
04F3 020500         =1  1552         JMP GSC_ERROR_XMIT_END
                    =1  1553
                    =1  1554     NOACK_ERROR:
                    =1  1555
04F6 30DE05         =1  1556         JNB NOACK,TCDT_ERROR                     ;see if error caused by
                    =1  1557                                                  ;NOACK
                    =1  1558
04F9 78D5           =1  1559         MOV ERROR_POINTER,#NOACK_COUNTER         ;load pointer with beginning
                    =1  1560                                                  ;address of NOACK counter
                    =1  1561
04FB 020500         =1  1562         JMP GSC_ERROR_XMIT_END
                    =1  1563
                    =1  1564     TCDT_ERROR:
                    =1  1565
04FE 78DB           =1  1566         MOV ERROR_POINTER,#TCDT_COUNTER          ;TCDT is only error left
                    =1  1567
                    =1  1568     GSC_ERROR_XMIT_END:
                    =1  1569
                    =1  1570     ;*******************************************************************
                    =1  1571     ; LOG FAILURE
                    =1  1572     ;*******************************************************************
                    =1  1573
0500 5175           =1  1574         CALL INCREMENT_COUNTER
                    =1  1575
                    =1  1576     ;*******************************************************************
                    =1  1577     ; RE-INITIALIZE DMA
                    =1  1578     ;*******************************************************************
                    =1  1579
0502 E52F           =1  1580         MOV A,BUFFER1_CONTROL
                    =1  1581
0504 540E           =1  1582         ANL A,#0EH                               ;mask off all bits except
                    =1  1583                                                  ;current buffer indicator
                    =1  1584
0506 B40012         =1  1585         CJNE A,#00,BUFFER1B_RELOAD               ;if current buffer is not 1A
                    =1  1586                                                  ;check for next buffer
                    =1  1587
```

270720-42

```
LOC  OBJ             LINE     SOURCE

0509 75A203      =1  1588         MOV SARLO,#LOW (BUF1A_STRT_ADDR)
050C 75A300      =1  1589         MOV SARHO,#HIGH (BUF1A_STRT_ADDR)
                 =1  1590                                          ;re-initialize source pointer
                 =1  1591                                          ;to BUF1A
                 =1  1592
050F 900000      =1  1593         MOV DPTR,#(BUF1A_STRT_ADDR) -3   ;location that holds BUF1A
                 =1  1594                                          ;byte count
                 =1  1595
0512 E0          =1  1596         MOVX A,@DPTR                     ;get byte count
                 =1  1597
0513 F5E2        =1  1598         MOV BCRLO,A
0515 75E300      =1  1599         MOV BCRHO,#0                     ;re-initialize byte counter
                 =1  1600                                          ;with number of bytes in BUF1A
                 =1  1601
0518 020554      =1  1602         JMP START_RETRANSMIT
                 =1  1603
                 =1  1604     BUFFER1B_RELOAD:
                 =1  1605
051B B40412      =1  1606         CJNE A,#04H,BUFFER1C_RELOAD      ;if current buffer is not 1B
                 =1  1607                                          ;check for next buffer
                 =1  1608
051E 75A283      =1  1609         MOV SARLO,#LOW (BUF1B_STRT_ADDR)
0521 75A300      =1  1610         MOV SARHO,#HIGH (BUF1B_STRT_ADDR)
                 =1  1611                                          ;re-initialize source pointer
                 =1  1612                                          ;to BUF1B
                 =1  1613
0524 900080      =1  1614         MOV DPTR,#(BUF1B_STRT_ADDR) -3   ;location that holds BUF1B
                 =1  1615                                          ;byte count
                 =1  1616
0527 E0          =1  1617         MOVX A,@DPTR                     ;get byte count
                 =1  1618
0528 F5E2        =1  1619         MOV BCRLO,A
052A 75E300      =1  1620         MOV BCRHO,#0                     ;re-initialize byte counter
                 =1  1621                                          ;with number of bytes in BUF1A
                 =1  1622
052D 020554      =1  1623         JMP START_RETRANSMIT
                 =1  1624
                 =1  1625     BUFFER1C_RELOAD:
                 =1  1626
0530 B40812      =1  1627         CJNE A,#08H,BUFFER1D_RELOAD      ;if current buffer is not 1C
                 =1  1628                                          ;check for next buffer
                 =1  1629
0533 75A203      =1  1630         MOV SARLO,#LOW (BUF1C_STRT_ADDR)
0536 75A301      =1  1631         MOV SARHO,#HIGH (BUF1C_STRT_ADDR)
                 =1  1632                                          ;re-initialize source pointer
                 =1  1633                                          ;to BUF1C
                 =1  1634
0539 900100      =1  1635         MOV DPTR,#(BUF1C_STRT_ADDR) -3   ;location that holds BUF1C
                 =1  1636                                          ;byte count
                 =1  1637
053C E0          =1  1638         MOVX A,@DPTR                     ;get byte count
                 =1  1639
053D F5E2        =1  1640         MOV BCRLO,A
053F 75E300      =1  1641         MOV BCRHO,#0                     ;re-initialize byte counter
                 =1  1642                                          ;with number of bytes in BUF1A
```

270720-43

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                    10/19/88   PAGE   28

LOC  OBJ            LINE     SOURCE

                =1  1643
0542 020554     =1  1644         JMP START_RETRANSMIT
                =1  1645
                =1  1646     BUFFER1D_RELOAD
                =1  1647
0545 75A283     =1  1648         MOV SARLO,#LOW (BUF1D_STRT_ADDR)
0548 75A301     =1  1649         MOV SARHO,#HIGH (BUF1D_STRT_ADDR)
                =1  1650                                            ;re-initialize source pointer
                =1  1651                                            ;to BUF1D
                =1  1652
054B 900180     =1  1653         MOV DPTR,#(BUF1D_STRT_ADDR) -3     ;location that holds BUF1D
                =1  1654                                            ;byte count
                =1  1655
054E E0         =1  1656         MOVX A,@DPTR                       ;get byte count
                =1  1657
054F F5E2       =1  1658         MOV BCRLO,A
0551 75E300     =1  1659         MOV BCRHO,#0                       ;re-initialize byte counter
                =1  1660                                            ;with number of bytes in BUF1A
                =1  1661
                =1  1662     START_RETRANSMIT:
                =1  1663
                =1  1664     ;*********************************************************************
                =1  1665     ; ENABLE TRANSMITTER AND DMA CHANNEL
                =1  1666     ;*********************************************************************
                =1  1667
0554 75D400     =1  1668         MOV TCDCNT,#0                      ;clear collision counter
                =1  1669
0557 D2D9       =1  1670         SETB TEN
                =1  1671
0559 30D9FD     =1  1672         JNB TEN,$                          ;wait until TEN is set (TEN
                =1  1673                                            ;will not be set if a
                =1  1674                                            ;transmissions CRC has not yet
                =1  1675                                            ;completed but TEN might be
                =1  1676                                            ;cleared before CRC completes)
                =1  1677
055C 439201     =1  1678         ORL DCONO,#01                      ;set GO bit
                =1  1679
055F D0D0       =1  1680         POP PSW
0561 D0E0       =1  1681         POP ACC
0563 D083       =1  1682         POP DPH
0565 D082       =1  1683         POP DPL                            ;SFRs that were saved
                =1  1684
0567 32         =1  1685         RETI
                =1  1686
                    1687 +1  $INCLUDE (RECVAL.SRC)
                =1  1688     GSC_VALID_REC:
                =1  1689
0568 C082       =1  1690         PUSH DPL
056A C083       =1  1691         PUSH DPH
056C C0E0       =1  1692         PUSH ACC
056E C0D0       =1  1693         PUSH PSW                           ;SFRs to save before servicing
                =1  1694                                            ;interrupt
                =1  1695
0570 71B0       =1  1696         CALL NEW_BUFFER2_IN                ;save byte count, select next
                =1  1697                                            ;GSC input buffer, setup next
```

270720-44

```
LOC  OBJ             LINE     SOURCE

                     =1  1698                                    ;destination address, and
                     =1  1699                                    ;setup new byte count
                     =1  1700
0572 439301          =1  1701         ORL DCON1,#01              ;set GO bit for DMA 1
                     =1  1702
0575 D2E9            =1  1703         SETB GREN                  ;enable receiver
                     =1  1704
0577 D0D0            =1  1705         POP PSW
0579 D0E0            =1  1706         POP ACC
057B D083            =1  1707         POP DPH
057D D082            =1  1708         POP DPL                    ;SFRs that were saved
                     =1  1709
057F 32              =1  1710         RETI
                         1711 +1  $INCLUDE (RECERR SRC)
                     =1  1712     GSC_ERROR_REC
                     =1  1713
0580 C082            =1  1714         PUSH DPL
0582 C083            =1  1715         PUSH DPH
0584 C0E0            =1  1716         PUSH ACC
0586 C0D0            =1  1717         PUSH PSW                   ;SFRs to save before servicing
                     =1  1718                                    ;interrupt
                     =1  1719
                     =1  1720
                     =1  1721     ;*********************************************************************
                     =1  1722     ; LOG ERROR TYPE
                     =1  1723     ;*********************************************************************
                     =1  1724
                     =1  1725     INC_ERROR_COUNT:
                     =1  1726     ;*********************************************************************
                     =1  1727     ; THIS ROUTINE INCREMENTS THE ERROR COUNT (UPTO 6 BYTES) FOR EACH TYPE
                     =1  1728     ; OF ERROR DETECTED BY HARDWARE
                     =1  1729     ;
                     =1  1730     ; BECAUSE OTHER ERROR BITS MAY BE SET WHEN OVR IS SET, OVR MUST BE TESTED
                     =1  1731     ; BEFORE AE OR CRCE.  ALSO, IN MOST APPLICATIONS AN ABORT MAY ALSO CAUSE
                     =1  1732     ; AN ALIGNMENT ERROR OR CRC ERROR, AND AN ALIGNMENT ERROR MAY CAUSE A CRC
                     =1  1733     ; ERROR, THE FOLLOWING SEQUENCE OF CHECKING ERROR BITS SHOULD BE FOLLOWED
                     =1  1734     ; TO GET AN  ACCURATE TALLY OF THE TYPES OF ERRORS THAT ARE OCCURRING
                     =1  1735     ;
                     =1  1736     ; COMBINATION OF ERROR BITS I HAVE SEEN:
                     =1  1737     ;   CRCE SET FOR BAD CRC
                     =1  1738     ;   RCABT AND AE SET FOR RCABT (ALIGNMENT ERROR MAY ALSO EXIST)
                     =1  1739     ;   AE AND CRCE SET FOR ALIGNMENT ERROR (CRC WAS BAD ALSO)
                     =1  1740     ;   OVR, AE, CRCE AND RFNE SET FOR OVR (THOUGH CRC IS GOOD AND NO AE)
                     =1  1741     ;
                     =1  1742     ;*********************************************************************
                     =1  1743
                     =1  1744     RCABT_CHECK:
0588 30EE07          =1  1745         JNB RCABT,OVR_CHECK        ;see if error caused by RCABT
                     =1  1746
058B 78F3            =1  1747         MOV ERROR_POINTER,#RCABT_COUNTER
                     =1  1748
058D 5175            =1  1749         CALL INCREMENT_COUNTER
                     =1  1750
058F 0205AA          =1  1751         JMP REC_ERROR_COUNT_END
                     =1  1752
```

270720-45

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                    10/19/88   PAGE   30

LOC  OBJ             LINE     SOURCE

                =1   1753     OVR_CHECK:
0592 30EF07     =1   1754         JNB OVR,CRC_CHECK                  ;see if error caused by OVR
                =1   1755
0595 78F9       =1   1756         MOV ERROR_POINTER,#OVR_COUNTER
                =1   1757
0597 5175       =1   1758         CALL INCREMENT_COUNTER
                =1   1759
0599 0205AA     =1   1760         JMP REC_ERROR_COUNT_END
                =1   1761
                =1   1762     CRC_CHECK:
059C 30EC07     =1   1763         JNB CRCE,AE_CHECK                  ;see if error caused by CRCE
                =1   1764
059F 78E7       =1   1765         MOV ERROR_POINTER,#CRCE_COUNTER
                =1   1766
05A1 5175       =1   1767         CALL INCREMENT_COUNTER
                =1   1768
05A3 0205AA     =1   1769         JMP REC_ERROR_COUNT_END
                =1   1770
                =1   1771     AE_CHECK:
05A6 78ED       =1   1772         MOV ERROR_POINTER,#AE_COUNTER      ;only error type left
                =1   1773
05A8 5175       =1   1774         CALL INCREMENT_COUNTER
                =1   1775
                =1   1776     REC_ERROR_COUNT_END:
                =1   1777     ;~
                =1   1778     ;this is not what I want to do probably.  I may need to fool with current
                =1   1779     ;active bit, addressing, byte count or who knows what????
                =1   1780
05AA 71B0       =1   1781         CALL NEW_BUFFER2_IN                ;say what this routine does
                =1   1782
05AC 439301     =1   1783         ORL DCON1,#01                      ;set GO bit for DMA1
                =1   1784
05AF D2E9       =1   1785         SETB GREN                          ;enable receiver
                =1   1786
05B1 D0D0       =1   1787         POP PSW
05B3 D0E0       =1   1788         POP ACC
05B5 D083       =1   1789         POP DPH
05B7 D082       =1   1790         POP DPL                            ;SFRs that were saved
                =1   1791
05B9 32         =1   1792         RETI
                =1   1793
                     1794 +1   $INCLUDE (LSCSERV.SRC)
                =1   1795     LSC_SERVICE:
                =1   1796
05BA C082       =1   1797         PUSH DPL
05BC C083       =1   1798         PUSH DPH
05BE C0E0       =1   1799         PUSH ACC
05C0 C0D0       =1   1800         PUSH PSW                           ;SFRs to save before servicing
                =1   1801                                            ;interrupt
                =1   1802
05C2 30980C     =1   1803         JNB RI,XMIT_LSC                    ;jump to LSC transmit service
                =1   1804                                            ;routine if RI is not set
                =1   1805
05C5 1205DD     =1   1806         CALL LSC_RECEIVE                   ;invoke LSC receiver server
                =1   1807
```

270720-46

```
LOC  OBJ            LINE     SOURCE

05C8 D0D0       =1  1808         POP PSW
05CA D0E0       =1  1809         POP ACC
05CC D083       =1  1810         POP DPH
05CE D082       =1  1811         POP DPL                        ;SFRs that were saved
                =1  1812
05D0 32         =1  1813         RETI                           ;return from interrupt
                =1  1814
                =1  1815     XMIT_LSC:
                =1  1816
05D1 1205FF     =1  1817         CALL LSC_XMIT                  ;invoke LSC transmit server
                =1  1818
05D4 D0D0       =1  1819         POP PSW
05D6 D0E0       =1  1820         POP ACC
05D8 D083       =1  1821         POP DPH
05DA D082       =1  1822         POP DPL                        ;SFRs that were saved
                =1  1823
05DC 32         =1  1824         RETI                           ;return from interrupt
                =1  1825
                =1  1826     LSC_RECEIVE:
                =1  1827
05DD C298       =1  1828         CLR RI                         ;clear receiver interrupt bit
                =1  1829
05DF 057F       =1  1830         INC IN_BYTE_COUNT              ;increment RAM location that
                =1  1831                                        ;counts the number of bytes
                =1  1832                                        ;input from LSC
                =1  1833
05E1 857B82     =1  1834         MOV DPL,LSC_INPUT_LOW
05E4 857A83     =1  1835         MOV DPH,LSC_INPUT_HIGH         ;get address where next byte
                =1  1836                                        ;received by LSC will be stored
                =1  1837
05E7 E599       =1  1838         MOV A,SBUF                     ;get oldest byte LSC has
                =1  1839                                        ;received
                =1  1840
05E9 F0         =1  1841         MOVX @DPTR,A                   ;store byte in buffer
                =1  1842
05EA A3         =1  1843         INC DPTR                       ;increment buffer address
                =1  1844
05EB 85827B     =1  1845         MOV LSC_INPUT_LOW,DPL
05EE 85837A     =1  1846         MOV LSC_INPUT_HIGH,DPH         ;store incremented address
                =1  1847
05F1 B40D0A     =1  1848         CJNE A,#CR,END_LSC_RECEIVE     ;initialize for next buffer
                =1  1849                                        ;if last character received
                =1  1850                                        ;was an ASCII carriage return
                =1  1851
05F4 057F       =1  1852         INC IN_BYTE_COUNT              ;increment RAM location that
                =1  1853                                        ;counts the number of bytes
                =1  1854                                        ;input from LSC
                =1  1855
05F6 740A       =1  1856         MOV A,#LINE_FEED               ;insert a line feed after the
                =1  1857                                        ;carraige return for GSC to
                =1  1858                                        ;transmit
                =1  1859
05F8 F0         =1  1860         MOVX @DPTR,A                   ;store byte in buffer
                =1  1861
05F9 5196       =1  1862         CALL NEW_BUFFER1_IN            ;setup for next buffer if
```

270720-47

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                    10/19/88   PAGE   32

LOC  OBJ            LINE     SOURCE

                    =1  1863                                    ;linefeed received
                    =1  1864
05FB 757F02         =1  1865         MOV IN_BYTE_COUNT,#02      ;2 needed for destination and
                    =1  1866                                    ;source address which do not
                    =1  1867                                    ;increment BYTE_COUNT when
                    =1  1868                                    ;loaded
                    =1  1869
                    =1  1870     END_LSC_RECEIVE:
                    =1  1871
05FE 22             =1  1872         RET
                    =1  1873
                    =1  1874     LSC_XMIT:
                    =1  1875
05FF D57508         =1  1876         DJNZ LSC_OUT_COUNTER,LSC_OUT_NEXT      ;continue outputting bytes
                    =1  1877                                                ;until counter reaches 0
                    =1  1878
0602 12062A         =1  1879         CALL CLR_ACTIVE_OUT        ;clear active buffer bit for
                    =1  1880                                    ;last buffer used
                    =1  1881
0605 C26E           =1  1882         CLR LSC_ACTIVE             ;indicate that LSC is no longer
                    =1  1883                                    ;trying to xmit a packet
                    =1  1884
                    =1  1885     LSC_XMIT_END:
                    =1  1886
0607 C299           =1  1887         CLR TI                     ;clear LSC xmit interrupt bit
                    =1  1888
0609 22             =1  1889         RET
                    =1  1890
                    =1  1891     LSC_OUT_NEXT:
                    =1  1892
060A 857782         =1  1893         MOV DPL,LSC_OUTPUT_LOW
060D 857683         =1  1894         MOV DPH,LSC_OUTPUT_HIGH    ;load DPTR with address of
                    =1  1895                                    ;next byte to xmit
                    =1  1896
0610 E0             =1  1897         MOVX A,@DPTR               ;get next byte
                    =1  1898
0611 F599           =1  1899         MOV SBUF,A                 ;load byte into LSC xmitter
                    =1  1900
0613 A3             =1  1901         INC DPTR                   ;increment LSC input address
                    =1  1902
0614 858277         =1  1903         MOV LSC_OUTPUT_LOW,DPL
0617 858376         =1  1904         MOV LSC_OUTPUT_HIGH,DPH    ;store incremented address
                    =1  1905
061A 80EB           =1  1906         JMP LSC_XMIT_END           ;return to main program
                    =1  1907
                        1908 +1  $INCLUDE (DMASERV.SRC)
                    =1  1909     DMA1_SERVICE:              ;to get to this point means that a
                    =1  1910                                ;message has been received which is
                    =1  1911                                ;longer than the maximum specified
                    =1  1912                                ;length - MAX_LENGTH (120)
061C C082           =1  1913         PUSH DPL
061E C083           =1  1914         PUSH DPH
0620 C0E0           =1  1915         PUSH ACC
0622 C0D0           =1  1916         PUSH PSW                   ;SFRs to save before servicing
                    =1  1917                                    ;interrupt
```

270720-48

```
LOC  OBJ             LINE     SOURCE

                =1   1918
0624 78E1       =1   1919         MOV ERROR_POINTER,#LONG_COUNTER
                =1   1920
0626 5175       =1   1921         CALL  INCREMENT_COUNTER
                =1   1922
0628 8080       =1   1923         JMP REC_ERROR_COUNT_END
                =1   1924
                =1   1925
                     1926 +1   $INCLUDE (LSCMGT.SRC)
                =1   1927     CLR_ACTIVE_OUT
                =1   1928
062A 207109     =1   1929         JB LSC_OUT_MSB,CLR_ACT_2B_2C          ;if LSC_OUT_MSB = 1B, buffer
                =1   1930                                               ;just emptied must be 2B or 2C
                =1   1931
                =1   1932     CLR_ACT_2A_2D
                =1   1933
062D 307003     =1   1934         JNB LSC_OUT_LSB,CLR_ACT_2D            ;if LSC_OUT = 00B, buffer just
                =1   1935                                               ;emptied is 2D
                =1   1936
                =1   1937     CLR_ACT_2A:
                =1   1938
0630 C277       =1   1939         CLR BUF2A_ACTIVE                      ;if LSC_OUT = 01B, buffer just
                =1   1940                                               ;emptied is 2A
                =1   1941
0632 22         =1   1942         RET
                =1   1943
                =1   1944     CLR_ACT_2D:
                =1   1945
0633 C274       =1   1946         CLR BUF2D_ACTIVE                      ;LSC_OUT = 00B
                =1   1947
0635 22         =1   1948         RET
                =1   1949
                =1   1950     CLR_ACT_2B_2C:
                =1   1951
0636 207003     =1   1952         JB LSC_OUT_LSB,CLR_ACT_2C             ;if LSC_OUT = 11B then buffer
                =1   1953                                               ;just empted must be 2C
                =1   1954
                =1   1955     CLR_ACT_2B:
                =1   1956
0639 C276       =1   1957         CLR BUF2B_ACTIVE                      ;if LSC_OUT = 10B, buffer just
                =1   1958                                               ;emptied must be 2B
                =1   1959
063B 22         =1   1960         RET
                =1   1961
                =1   1962     CLR_ACT_2C:
                =1   1963
063C C275       =1   1964         CLR BUF2C_ACTIVE                      ;LSC_OUT = 11B
                =1   1965
063E 22         =1   1966         RET
                =1   1967
                     1968
                     1969     END
```

270720-49

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                    10/19/88   PAGE   34


XREF SYMBOL TABLE LISTING
---- ------ ----- -------


N A M E                   T Y P E   V A L U E       ATTRIBUTES AND REFERENCES

AC . . . . . .              NUMB    00D6H   A         76#
ACC . . . . . .             NUMB    00E0H   A         15# 1439 1523 1540 1681 1692 1706 1716 1788 1799 1809 1820 1915
ADDRESS_DETERMINATION . .   C ADDR  0243H   A         383 499#
ADR0 . . . . . .            NUMB    0095H   A         39# 469
ADR1 . . . . . .            NUMB    00A5H   A         43#
ADR2 . . . . . .            NUMB    00B5H   A         47#
ADR3 . . . . . .            NUMB    00C5H   A         52#
AE_CHECK . . . . . .        C ADDR  05A6H   A         1763 1771#
AE_COUNTER . . . . . .      D ADDR  00EDH   A         229# 232 1772
AE . . . . . .              NUMB    00EDH   A         158#
AMSK0 . . . . . .           NUMB    00D5H   A         57#
AMSK1 . . . . . .           NUMB    00E5H   A         62#
B . . . . . .               NUMB    00F0H   A         16#
BAUD . . . . . .            NUMB    0094H   A         38# 442
BCRH0 . . . . . .           NUMB    00E3H   A         60# 910 940 976 1006 1599 1620 1641 1659
BCRH1 . . . . . .           NUMB    00F3H   A         65# 460 1263
BCRL0 . . . . . .           NUMB    00E2H   A         59# 907 937 973 1003 1598 1619 1640 1658
BCRL1 . . . . . .           NUMB    00F2H   A         64# 461 1097 1138 1197 1237 1264
BKOFF . . . . . .           NUMB    00C4H   A         51#
BUF1A_ACTIVE . . . . . .    B ADDR  002FH.4 A         297# 300 397 696 826 898 1479
BUF1A_STRT_ADDR . . . . .   NUMB    0003H   A         174# 542 543 672 673 863 864 902 1588 1589 1593
BUF1B_ACTIVE . . . . . .    B ADDR  002FH.5 A         294# 297 400 668 743 928 1495
BUF1B_STRT_ADDR . . . . .   NUMB    0083H   A         178# 705 706 719 720 932 1609 1610 1614
BUF1C_ACTIVE . . . . . .    B ADDR  002FH.6 A         291# 294 403 715 808 964 1502
BUF1C_STRT_ADDR . . . . .   NUMB    0103H   A         182# 752 753 784 785 968 1630 1631 1635
BUF1D_ACTIVE . . . . . .    B ADDR  002FH.7 A         288# 291 406 780 854 994 1471
BUF1D_STRT_ADDR . . . . .   NUMB    0183H   A         186# 817 818 830 831 998 1648 1649 1653
BUF2A_ACTIVE . . . . . .    B ADDR  002EH.7 A         316# 319 409 1102 1220 1293 1939
BUF2A_STRT_ADDR . . . . .   NUMB    0201H   A         190# 471 472 1084 1085 1251 1252 1300
BUF2B_ACTIVE . . . . . .    B ADDR  002EH.6 A         319# 322 412 1080 1143 1322 1957
BUF2B_STRT_ADDR . . . . .   NUMB    0281H   A         193# 1111 1112 1125 1126 1329
BUF2C_ACTIVE . . . . . .    B ADDR  002EH.5 A         322# 325 415 1121 1202 1357 1964
BUF2C_STRT_ADDR . . . . .   NUMB    0301H   A         196# 1152 1153 1184 1185 1364
BUF2D_ACTIVE . . . . . .    B ADDR  002EH.4 A         325# 328 418 1180 1242 1386 1946
BUF2D_STRT_ADDR . . . . .   NUMB    0381H   A         199# 1211 1212 1224 1225 1393
BUFFER1_CONTROL . . . . .   D ADDR  002FH   A         280# 530 1580
BUFFER1_START . . . . . .   C ADDR  012CH   A         397 400 403 406 423#
BUFFER1B_RELOAD . . . . .   C ADDR  051BH   A         1585 1604#
BUFFER1C_RELOAD . . . . .   C ADDR  0530H   A         1606 1625#
BUFFER1D_RELOAD . . . . .   C ADDR  0545H   A         1627 1646#
BUFFER2_CONTROL . . . . .   D ADDR  002EH   A         283# 535
BUFFER2_START . . . . . .   C ADDR  0131H   A         409 412 415 418 430#
BUFFERS_1_FULL . . . . . .  C ADDR  02E2H   A         668 715 759# 780 826
BUFFERS_2_FULL . . . . . .  C ADDR  03F2H   A         1080 1121 1159# 1180 1220
CLEAR_ACTIVE_1A . . . . .   C ADDR  04C4H   A         1459 1477#
CLEAR_ACTIVE_1B_1C . . . .  C ADDR  04CBH   A         1454 1488#
CLEAR_ACTIVE_1B . . . . .   C ADDR  04CEH   A         1493#
CLEAR_ACTIVE_1C . . . . .   C ADDR  04D3H   A         1490 1500#
CLEAR_ACTIVE_1D . . . . .   C ADDR  04BCH   A         1462#
CLEAR_ACTIVE_BUFFER . . .   C ADDR  04B6H   A         1452#
CLR_ACT_2A_2D . . . . . .   C ADDR  062DH   A         1932#
```

270720-50

```
N A M E                  T Y P E  V A L U E       ATTRIBUTES AND REFERENCES

CLR_ACT_2A . . . . . .   C ADDR   0630H   A       1937#
CLR_ACT_2B_2C. . . . .   C ADDR   0636H   A       1929 1950#
CLR_ACT_2B . . . . . .   C ADDR   0639H   A       1955#
CLR_ACT_2C . . . . . .   C ADDR   063CH   A       1952 1962#
CLR_ACT_2D . . . . . .   C ADDR   0633H   A       1934 1944#
CLR_ACTIVE_OUT . . . .   C ADDR   062AH   A       1879 1927#
COUNTER_CLEAR. . . . .   C ADDR   026EH   A       554# 559
COUNTER_OVERFLOW . . .   C ADDR   0282H   A       583 594#
CR . . . . . .             NUMB   000DH   A       204# 1848
CRC_CHECK. . . . . .     C ADDR   059CH   A       1754 1762#
CRCE_COUNTER . . . . .   D ADDR   00E7H   A       232# 235 1765
CRCE . . . . . .           NUMB   00ECH   A       159# 1763
CY . . . . . .             NUMB   00D7H   A       75#
DARH0. . . . . .           NUMB   00C3H   A       50#
DARH1. . . . . .           NUMB   00D3H   A       55# 476 1259
DARL0. . . . . .           NUMB   00C2H   A       49# 453
DARL1. . . . . .           NUMB   00D2H   A       54# 475 1258
DCON0. . . . . .           NUMB   0092H   A       36# 455 1028 1536 1678
DCON1. . . . . .           NUMB   0093H   A       37# 464 1701 1783
DMA. . . . . .             NUMB   00D8H   A       153# 451
DMA1_DONE. . . . . .     C ADDR   0053H   A       375#
DMA1_SERVICE . . . . .   C ADDR   061CH   A       376 1909#
DPH. . . . . .             NUMB   0083H   A       19# 673 720 785 831 916 946 982 1012 1085 1126 1185 1225 1422
                                                   1438 1524 1539 1682 1691 1707 1715 1789 1798 1810 1821 1835 1846
                                                   1894 1904 1914
DPL. . . . . .             NUMB   0082H   A       18# 672 719 784 830 915 945 981 1011 1084 1125 1184 1224 1421
                                                   1437 1525 1538 1683 1690 1708 1714 1790 1797 1811 1822 1834 1845
                                                   1893 1903 1913
EA . . . . . .             NUMB   00AFH   A       94# 523
EDMA0. . . . . .           NUMB   00CAH   A       133#
EDMA1. . . . . .           NUMB   00CCH   A       131# 521
EGSRE. . . . . .           NUMB   00C9H   A       134# 517
EGSRV. . . . . .           NUMB   00C8H   A       135# 515
EGSTE. . . . . .           NUMB   00CDH   A       130# 1026 1449
EGSTV. . . . . .           NUMB   00CBH   A       132# 1023 1447
END_CLEAR_ACTIVE_OUT . . C ADDR   04D5H   A       1464 1475 1486 1498 1507#
END_LSC_RECEIVE. . . . . C ADDR   05FEH   A       1848 1870#
ERROR_POINTER. . . . . .   REG    R0              209# 573 577 579 596 598 600 602 604 606 608 610 612 614 616 618
                                                   1549 1559 1566 1747 1756 1765 1772 1919
ES . . . . . .             NUMB   00ACH   A       95# 519
ET0. . . . . .             NUMB   00A9H   A       98#
ET1. . . . . .             NUMB   00ABH   A       96#
EX0. . . . . .             NUMB   00A8H   A       99#
EX1. . . . . .             NUMB   00AAH   A       97#
F0 . . . . . .             NUMB   00D5H   A       77#
FIRST_GSC_OUT. . . . . . B ADDR   002DH.7 A       344# 346 481 1464 1482
GENERIC_INIT . . . . . . C ADDR   025BH   A       390 528#
GMOD . . . . . .           NUMB   0084H   A       34# 444
GREN . . . . . .           NUMB   00E9H   A       162# 479 1703 1785
GSC_BAUD_RATE. . . . . .   NUMB   0000H   A       166# 442
GSC_DEST_ADDR. . . . . . D ADDR   007DH   A       257# 260 508 685 732 797 843
GSC_ERROR_REC. . . . . . C ADDR   0580H   A       364 1712#
GSC_ERROR_XMIT_END . . . C ADDR   0500H   A       1552 1562 1568#
GSC_ERROR_XMIT . . . . . C ADDR   04E3H   A       372 1530#
GSC_IN_2A. . . . . .     C ADDR   0417H   A       1175 1218#
```

270720-51

```
MCS-51 MACRO ASSEMBLER    APPNOT1                                          10/19/88   PAGE   36

N A M E                   T Y P E   V A L U E       ATTRIBUTES AND REFERENCES

GSC_IN_2B. . . . . .      C ADDR    03B6H    A        1076#
GSC_IN_2C. . . . . .      C ADDR    03D4H    A        1073 1119#
GSC_IN_2D_2A . . . . .    C ADDR    03F6H    A        1069 1173#
GSC_IN_2D. . . . . .      C ADDR    03F9H    A        1178#
GSC_IN_LSB . . . . . .    B ADDR    002EH. 2 A        332# 336 1073 1108 1148 1175 1207 1247
GSC_IN_MSB . . . . . .    B ADDR    002EH. 3 A        328# 332 1069 1107 1149 1208 1248
GSC_INIT . . . . . .      C ADDR    0200H    A        386 440#
GSC_INPUT_HIGH . . . .    D ADDR    0078H    A        269# 273 472 476 1112 1153 1212 1252 1259
GSC_INPUT_LOW. . . . .    D ADDR    0079H    A        268# 269 471 475 1111 1152 1211 1251 1258
GSC_OUT_1A . . . . . .    C ADDR    033EH    A        895#
GSC_OUT_1B . . . . . .    C ADDR    0358H    A        892 925#
GSC_OUT_1C_1D. . . . .    C ADDR    0372H    A        889 955#
GSC_OUT_1C . . . . . .    C ADDR    0375H    A        961#
GSC_OUT_1D . . . . . .    C ADDR    038FH    A        957 991#
GSC_OUT_LSB. . . . . .    B ADDR    002FH. 2 A        304# 308 892 920 950 957 986 1016 1459 1490
GSC_OUT_MSB. . . . . .    B ADDR    002FH. 3 A        300# 304 889 919 949 985 1015 1454
GSC_REC_ERROR. . . . .    C ADDR    0033H    A        363#
GSC_REC_VALID. . . . .    C ADDR    002BH    A        359#
GSC_SRC_ADDR . . . . .    D ADDR    007CH    A        260# 263 469 503 692 739 804 850
GSC_VALID_REC. . . . .    C ADDR    0568H    A        360 1688#
GSC_VALID_XMIT . . . .    C ADDR    04AAH    A        368 1435#
GSC_XMIT_ERROR . . . .    C ADDR    004BH    A        371#
GSC_XMIT_VALID . . . .    C ADDR    0043H    A        367#
HABEN. . . . . .          NUMB      00E8H    A        163#
IE . . . . . .            NUMB      00A8H    A        27#
IE0. . . . . .            NUMB      0089H    A        90#
IE1. . . . . .            NUMB      008BH    A        88#
IEN1 . . . . . .          NUMB      00C8H    A        53#
IFS_PERIOD . . . . . .    NUMB      0014H    A        171# 447
IFS. . . . . .            NUMB      00A4H    A        42# 447
IN_BYTE_COUNT. . . . .    D ADDR    007FH    A        249# 253 547 677 724 789 835 1830 1852 1865
INC_COUNT_LOOP . . . .    C ADDR    0278H    A        571# 581
INC_ERROR_COUNT. . . .    C ADDR    0588H    A        1725#
INCREMENT_COUNTER. . .    C ADDR    0275H    A        565# 1574 1749 1758 1767 1774 1921
INITIALIZATION . . . .    C ADDR    0100H    A        353 379#
INT0 . . . . . .          NUMB      00B2H    A        114#
INT1 . . . . . .          NUMB      00B3H    A        113#
INTERRUPT_ENABLE . . .    C ADDR    0250H    A        393 513#
IP . . . . . .            NUMB      00B8H    A        28#
IPN1 . . . . . .          NUMB      00F8H    A        68#
IRET . . . . . .          C ADDR    032EH    A        761 872# 1161
IT0. . . . . .            NUMB      0088H    A        91#
IT1. . . . . .            NUMB      008AH    A        89#
LINE_FEED. . . . . .      NUMB      000AH    A        207# 1856
LNI. . . . . .            NUMB      00DFH    A        146#
LONG_COUNTER . . . . .    D ADDR    00E1H    A        235# 239 1919
LSC_ACTIVE . . . . . .    B ADDR    002DH. 6 A        346# 539 1271 1281 1297 1326 1361 1390 1882
LSC_BAUD_RATE. . . . .    NUMB      00FCH    A        168# 487
LSC_IN_1A. . . . . .      C ADDR    030DH    A        775 824#
LSC_IN_1B. . . . . .      C ADDR    029CH    A        664#
LSC_IN_1C. . . . . .      C ADDR    02BFH    A        661 713#
LSC_IN_1D_1A . . . . .    C ADDR    02E7H    A        657 773#
LSC_IN_1D. . . . . .      C ADDR    02EAH    A        778#
LSC_IN_LSB . . . . . .    B ADDR    002FH. 0 A        312# 316 661 702 748 775 813 859
LSC_IN_MSB . . . . . .    B ADDR    002FH. 1 A        308# 312 657 701 749 814 860
```

270720-52

NAME	TYPE	VALUE	ATTRIBUTES AND REFERENCES
LSC_INIT	C ADDR	0234H A	388 486#
LSC_INPUT_HIGH	D ADDR	007AH A	264# 268 543 706 753 818 864 1835 1846
LSC_INPUT_LOW	D ADDR	007BH A	263# 264 542 705 752 817 863 1834 1845
LSC_OUT_2A	C ADDR	044EH A	1290#
LSC_OUT_2B	C ADDR	0462H A	1287 1319#
LSC_OUT_2C_2D	C ADDR	0476H A	1284 1348#
LSC_OUT_2C	C ADDR	0479H A	1354#
LSC_OUT_2D	C ADDR	048DH A	1350 1383#
LSC_OUT_COUNTER	D ADDR	0075H A	277# 1305 1308 1334 1337 1369 1372 1398 1401 1876
LSC_OUT_LSB	B ADDR	002EH.0 A	340# 344 1287 1314 1343 1350 1378 1407 1934 1952
LSC_OUT_MSB	B ADDR	002EH.1 A	336# 340 1284 1313 1342 1377 1406 1929
LSC_OUT_NEXT	C ADDR	060AH A	1876 1891#
LSC_OUTPUT_HIGH	D ADDR	0076H A	274# 277 1422 1894 1904
LSC_OUTPUT_LOW	D ADDR	0077H A	273# 274 1421 1893 1903
LSC_RECEIVE	C ADDR	05DDH A	1806 1826#
LSC_SERVICE	C ADDR	05BAH A	356 1795#
LSC_XMIT_END	C ADDR	0607H A	1885# 1906
LSC_XMIT_IN_PROGRESS	C ADDR	0442H A	1276# 1281
LSC_XMIT	C ADDR	05FFH A	1817 1874#
MAIN	C ADDR	0112H A	395# 421 428 436
MAX_LENGTH	NUMB	0078H A	213# 461 1091 1132 1191 1231 1264
MYSLOT	NUMB	00F5H A	67#
NEW_BUF1_IN_END	C ADDR	032DH A	710 757 822 868#
NEW_BUF2_IN_END	C ADDR	0432H A	1116 1157 1216 1256#
NEW_BUFFER1_IN	C ADDR	0296H A	624# 770 1862
NEW_BUFFER1_OUT	C ADDR	032FH A	425 875# 1514
NEW_BUFFER2_IN	C ADDR	03B0H A	1036# 1170 1696 1781
NEW_BUFFER2_OUT	C ADDR	043FH A	432 1269#
NEXT_LOCATION	D ADDR	00CFH A	245# 552
NOACK_COUNTER	D ADDR	00D5H A	242# 245 1559
NOACK_ERROR	C ADDR	04F6H A	1546 1554#
NOACK	NUMB	00DEH A	147# 1556
NOTHING_FOR_GSC	C ADDR	03AFH A	882 898 928 964 994 1030#
NOTHING_FOR_LSC	C ADDR	04A9H A	1277 1293 1322 1357 1386 1429#
OUT_BYTE_COUNT	D ADDR	007EH A	253# 257
OV	NUMB	00D2H A	80#
OVR_CHECK	C ADDR	0592H A	1745 1753#
OVR_COUNTER	D ADDR	00F9H A	223# 226 1756
OVR	NUMB	00EFH A	156# 1754
P	NUMB	00D0H A	81#
P0	NUMB	0080H A	10#
P1	NUMB	0090H A	11# 501 506
P2	NUMB	00A0H A	12#
P3	NUMB	00B0H A	13#
P4	NUMB	00C0H A	48# 503 508
PCON	NUMB	0087H A	20#
PDMA0	NUMB	00FAH A	141#
PDMA1	NUMB	00FCH A	139#
PGSRE	NUMB	00F9H A	142#
PGSRV	NUMB	00F8H A	143#
PGSTE	NUMB	00FDH A	138#
PGSTV	NUMB	00FBH A	140#
PRBS	NUMB	00E4H A	61#
PS	NUMB	00BCH A	102#
PSW	NUMB	00D0H A	14# 1440 1522 1541 1680 1693 1705 1717 1787 1800 1808 1819 1916

270720-53

MCS-51 MACRO ASSEMBLER APPNOT1 10/19/88 PAGE 38

NAME	TYPE	VALUE		ATTRIBUTES AND REFERENCES
PTO	NUMB	00B9H	A	105#
PT1	NUMB	00BBH	A	103#
PXO	NUMB	00B8H	A	106#
PX1	NUMB	00BAH	A	104#
RBB	NUMB	009AH	A	124#
RCABT_CHECK	C ADDR	0588H	A	1744#
RCABT_COUNTER	D ADDR	00F3H	A	226# 229 1747
RCABT	NUMB	00EEH	A	157# 1745
RD	NUMB	00B7H	A	109#
RDN	NUMB	00EBH	A	160#
REC_ERROR_COUNT_END	C ADDR	05AAH	A	1751 1760 1769 1776# 1923
REN	NUMB	009CH	A	122#
RFIFO	NUMB	00F4H	A	66# 458
RFNE	NUMB	00EAH	A	161#
RI	NUMB	0098H	A	126# 1803 1828
RSO	NUMB	00D3H	A	79#
RS1	NUMB	00D4H	A	78#
RSTAT	NUMB	00E8H	A	63#
RXD	NUMB	00B0H	A	116#
SARHO	NUMB	00A3H	A	41# 916 946 982 1012 1589 1610 1631 1649
SARH1	NUMB	00B3H	A	45#
SARLO	NUMB	00A2H	A	40# 915 945 981 1011 1588 1609 1630 1648
SARL1	NUMB	00B2H	A	44# 458
SBUF	NUMB	0099H	A	30# 1838 1899
SCON	NUMB	0098H	A	29# 492
SECOND_LSC_CHECK	C ADDR	0445H	A	1271 1280#
SECOND_TEN_CHECK	C ADDR	0335H	A	877 885#
SLOTTM	NUMB	00B4H	A	46#
SMO	NUMB	009FH	A	119#
SM1	NUMB	009EH	A	120#
SM2	NUMB	009DH	A	121#
SP	NUMB	0081H	A	17# 381
STACK_OFFSET	NUMB	0080H	A	202# 381
START_GSC_OUT	C ADDR	03A6H	A	923 953 989 1019#
START_LSC_OUT	C ADDR	049EH	A	1317 1346 1381 1410#
START_RETRANSMIT	C ADDR	0554H	A	1602 1623 1644 1662#
START	C ADDR	0000H	A	351#
TO	NUMB	00B4H	A	112#
T1	NUMB	00B5H	A	111#
TBB	NUMB	009BH	A	123#
TCDCNT	NUMB	00D4H	A	56# 449 1520 1668
TCDT_COUNTER	D ADDR	00DBH	A	239# 242 1566
TCDT_ERROR	C ADDR	04FEH	A	1556 1564#
TCDT	NUMB	00DCH	A	149#
TCON	NUMB	0088H	A	21#
TDN	NUMB	00DBH	A	150#
TEN	NUMB	00D9H	A	152# 877 886 1021 1670 1672
TFO	NUMB	008DH	A	86#
TF1	NUMB	008FH	A	84#
TFIFO	NUMB	0085H	A	35# 453
TFNF	NUMB	00DAH	A	151#
THO	NUMB	008CH	A	25#
TH1	NUMB	008DH	A	26# 487
TI	NUMB	0099H	A	125# 1425 1887
TLO	NUMB	008AH	A	23#

270720-54

```
N A M E                    T Y P E   V A L U E       ATTRIBUTES AND REFERENCES

TL1. . . . . .               NUMB     008BH    A        24#
TMOD . . . . . .             NUMB     0089H    A        22# 489 490
TR0. . . . . .               NUMB     008CH    A        87#
TR1. . . . . .               NUMB     008EH    A        85# 495
TRANSMISSION_IN_PROGRESS  C ADDR      0332H    A        881# 886
TSTAT. . . . . .             NUMB     00D8H    A        58#
TXD. . . . . .               NUMB     00B1H    A        115#
UR_COUNTER . . . . . .    D ADDR      00FFH    A        219# 223 1549
UR_ERROR . . . . . .      C ADDR      04EEH    A        1544#
UR . . . . . .               NUMB     00DDH    A        148# 1546
WR . . . . . .               NUMB     00B6H    A        110#
XMIT_LSC . . . . . .      C ADDR      05D1H    A        1803 1815#


REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

270720-55

APPENDIX B
TAKING CONTROL OF THE BACKOFF ALGORITHM

There is a method that allows the user to take control of the backoff process. This method will only work when normal or alternate backoff modes are selected. It will not work in DCR mode. This method works by loading TCDCNT with 80H. Then on the first collision, TCDCNT will overflow, aborting the transmission and causing a transmission error to occur. It is in the error routine where the user takes control. Some of the modifications that have been tested are:

1) Extending the number of retransmissions—this was accomplished by counting the number of attempted transmissions in a user implemented counter. When the number of collisions grew too big, the transmissions were aborted and an error flag set.
2) Extending the number of time slots available—to implement this, it was required that the time slots be simulated using one of the timers. Then by reading the PRBS multiple times and ANDing each read of the PRBS with a masking register, the number of time slots could be extended to randomly fall within any range selected by the user. Once the slot time was determined, the resulting value was multiplied by the selected time slot with the appropriate value loaded into the timer registers and the timer started. When the timer expired, the transmission was re-attempted. For very large delays, multiple timer overflows were required and a loop counter used. This also allowed time slots larger than 255 bit times to be used.

Other modifications the user may wish to implement would be to use some kind of token passing scheme when collisions occur or instead of randomly assigning slot times, assign pre-determined time slots to each station.

If the user decides to implement some kind of scheme such as these there are several factors the user must be aware of. These are:

1) When TCDCNT overflows, it will still contain either 0 or 1 and these many time slots must expire before the GSC will begin transmissions again. Even if the transmitter is disabled and re-enabled the GSC still goes through the standard backoff algorithm. This means the user should program the slot time to 01 to minimize the amount of time until the GSC hardware will allow another transmission to begin.
2) Due to the amount of software required to implement any of these suggestions, most will not work at the same speed the internal hardware is capable of. For this reason, running at maximum baud rates with minimum IFS will probably not work.
3) There is no real time indication to the user that the GSC thinks it is in a backoff algorithm, if the GSC is currently receiving data, or when a collision is detected. These, and possibly other factors not apparent at the time this application note was written, must be considered whenever the user tries to modify the hardware based backoff algorithm with software.

APPENDIX C
REFERENCES

1. ISO (1979) Data Communication—High-Level Data Link Control Procedures—Frame Structure, ISO 3309.
2. ANSI/IEEE (1985) Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications, ANSI/IEEE Std 802.3.